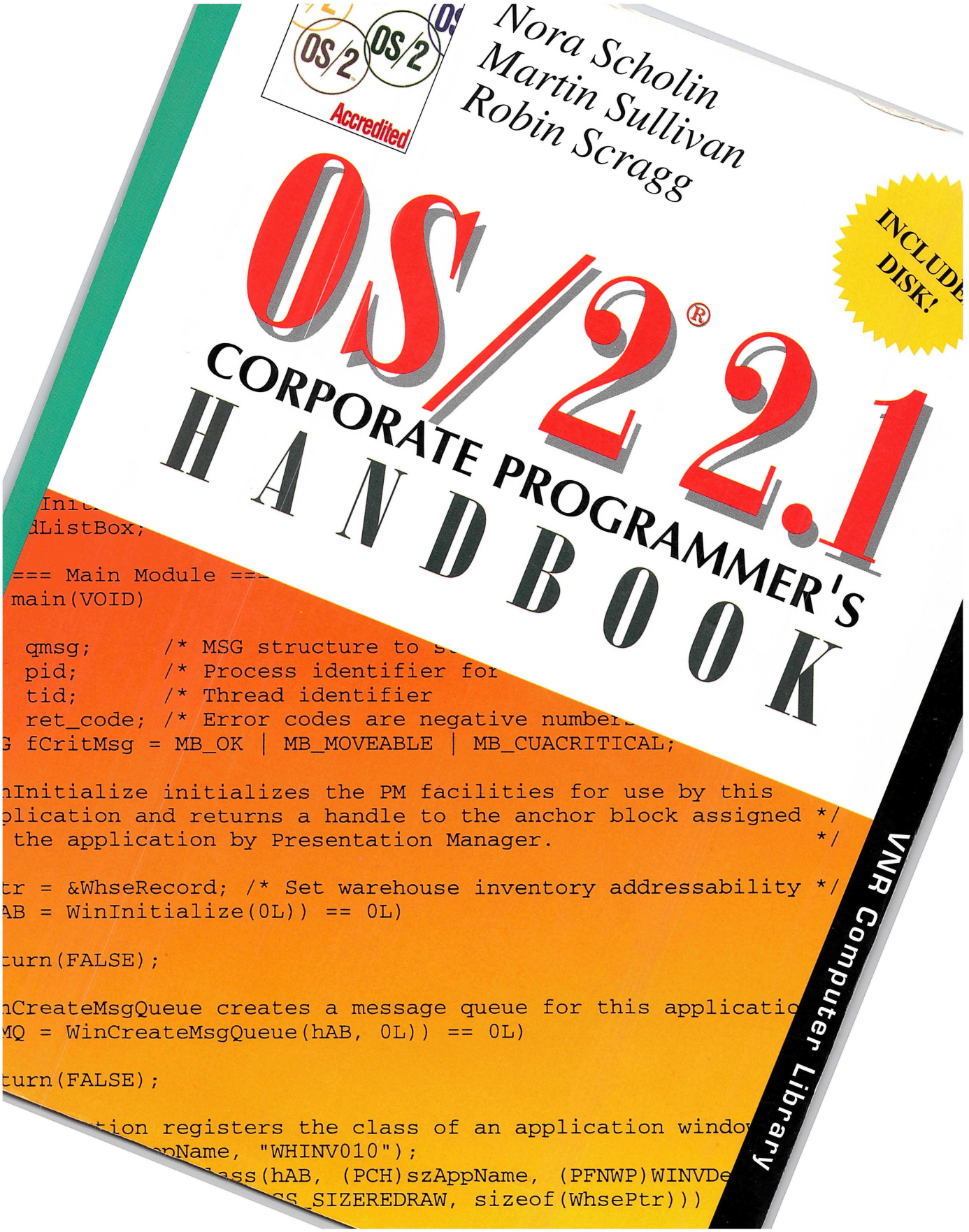

OS/2
OS/2
Accredited
Nora Scholin
Martin Sullivan
Robin Scragg
INCLUDES DISK!
OS/2® 2.1
CORPORATE PROGRAMMER'S
HANDBOOK
dListBox;
=== Main Module ===
main(VOID)
qmsg; /* MSG structure to
pid; /* Process identifier for
tid; /* Thread identifier
ret_code; /* Error codes are negative numbers
fCritMsg = MB_OK | MB_MOVEABLE | MB_CUACRITICAL;
nInitialize initializes the PM facilities for use by this
plication and returns a handle to the anchor block assigned */
the application by Presentation Manager. */
tr = &WhseRecord; /* Set warehouse inventory addressability */
AB = WinInitialize(0L)) == 0L)
turn(FALSE);
nCreateMsgQueue creates a message queue for this applicatio
MQ = WinCreateMsgQueue(hAB, 0L)) == 0L)
turn(FALSE);
tion registers the class of an application windo
pName, "WHINV010");
lass(hAB, (PCH)szAppName, (PFNWP)WINVDe
S_SIZEREDRAW, sizeof(WhsePtr)))
VNR Computer Library

OS/2 2.1

Corporate Programmer's Handbook

by Nora Scholin, Martin Sullivan, and Robin Scragg

VAN NOSTRAND REINHOLD
New York

Copyright © 1993 by Van Nostrand Reinhold

Library of Congress Catalog Card Number 93-3025
ISBN 0-442-01598-4

All rights reserved. No part of this work covered by the copyright hereon may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems—without the written permission of the publisher.

I(T)P Van Nostrand Reinhold is an International Thomson Publishing company.
ITP logo is a trademark under license.

Printed in the United States of America

Van Nostrand Reinhold
115 Fifth Avenue
New York, NY 10003

International Thomson Publishing GmbH
Konigswinterer Str. 518
5300 Bonn 3
Germany

International Thomson Publishing
Berkshire House,168-173
High Holborn, London WC1V 7AA
England

International Thomson Publishing Asia
38 Kim Tian Rd., #0105
Kim Tian Plaza
Singapore 0316

Thomas Nelson Australia
102 Dodds Street
South Melbourne 3205
Victoria, Australia

International Thomson Publishing Japan
Kyowa Building, 3F
2-2-1 Hirakawacho
Chiyada-Ku, Tokyo 102
Japan

Nelson Canada
1120 Birchmount Road
Scarborough, Ontario
M1K 5G4, Canada

16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging in Publication Data

Scholin, Nora, 1952-
The OS/2 2.1 corporate programmer's handbook / Nora Scholin,
Martin Sullivan, Robin Scragg.
p. cm.
ISBN 0-442-01598-4
1. Operating systems (Computers) 2. OS/2 (Computer file)
I. Sullivan, Martin (Martin C.), 1949- . II. Scragg, Robin.
III. Title.
QA76.76.O63S385 1993
005.4'469—dc20 93-3025
CIP

To my family, friends, and publishers, who patiently waited while I finished this book, and for GMJ, who showed me what's really important.

Nora Scholin

Notices

This book is not intended to replace IBM or other companies documentation in determining the specifications or capabilities of any products referenced within.

The authors and publishers of this book make no warranty of any kind, expressed or implied, with regard to the code and documentation contained in this book and accompanying diskette. You are responsible for selecting all your configurations and applications of hardware and software. The authors and publishers shall not be liable in any event for any loss or damages in connection with, or arising out of, use of this book or the companion diskette.

Information is this book is subject to change without notice.

IBM Trademarks

The following terms are trademarks or service marks of IBM Corporation:

- C Set/2
- Common User Access
- CUA
- IBM
- Information Presentation Facility
- Micro Channel
- Multimedia Presentation Manager/2
- OS/2
- Presentation Manager
- SAA
- Systems Application Architecture
- WIN-OS/2
- WorkFrame/2
- Workplace Shell
- XGA

Other Trademarks

The following terms are trademarks or service marks of other companies:

Apple	Apple Computer, Inc.
Borland C++	Borland International, Inc.
DEC	Digital Equipment Corporation
DeskJet	Hewlett-Packard Company
Epson	Seiko Epson Kabushiki Kaisha
Excel	Microsoft Corporation
HP	Hewlett-Packard Company
Intel	Intel Corporation
LaserJet	Hewlett-Packard Company
Lotus	Lotus Development Corporation
Macintosh	Apple Computer, Inc.
Microsoft	Microsoft Corporation
MS	Microsoft Corporation
NetWare	Novell, Inc.
PageMaker	Aldus Corporation
PaintJet	Hewlett-Packard Company
PCMCIA	Personal Computer Memory Card International Association
Pentium	Intel Corporation
PostScript	Adobe Systems Incorporated
UNIX	UNIX System Laboratories, Inc.
VAX	Digital Equipment Corporation
Windows	Microsoft Corporation

Table of Contents

Notices v

IBM Trademarks v
Other Trademarks vi

Preface xiv

Who Should Use This Book xiv
How This Book Is Organized xv
Conventions Used in This Book xv
About the Companion Diskette xvi
Special Thanks xvi

Part I. Tips on Installing and Using OS/2 2.1 1

Chapter 1.
Key Features of OS/2 2.1 3

CD-ROM Installation 3
Improved and Additional OS/2 2.1 Support 3
  Additional Device Driver Support 3
  Windows 3.1 Support 4
  MMPM/2 Support 4
  Dual Thread Support 4
Fewer Performance and Memory Restrictions 4
  32-Bit Flat Memory Model 4
  32-Bit Graphical Environment 5
Multiple Virtual DOS Machine 5
Workplace Shell 5
  OS/2 Desktop 6
  Command Prompts and System Folders 6
  Settings Notebook 7
  Clipboard Interactivity 7
Integrated LAN Resources 7
Improved Online Help 8

Chapter 2.
Tips on Installing OS/2 2.1 9

Before Installation: Back Up Your Applications 9
What To Do First 10
Microprocessor Requirements 11
System Memory Requirements 11

Hard Disk Space Requirements 12
Installing OS/2 2.1 from Diskette: Step 1 13
Installing Boot Manager 13
Example of Partitioning and Boot Manager 19
Installing the High Performance File System 21
Disk Drive Considerations 22
Installing OS/2 2.1 from Diskette: Step 2 22
OS/2 Setup and Installation Screen 24
Moving the Swapper File 26
Installing the IBM OS/2 2.x Toolkit 28
Installing the IBM C Set/2 Compiler 30

Chapter 3.
Tips on Compiling Your OS/2 Application 33

Using Compilers in OS/2 2.1 33
Compiling and Optimizing Your Application 34
Optimizing with the IBM C Set/2 Compiler 35
Optimizing with the Borland C++ Compiler 35
Invoking the Linker Facility 36
Using Dynamic Link Libraries 36
Using Header Files 37
Using Run-Time Libraries 38
Accessing DLL Functions from Your Application 39
Creating a Development Environment 40
Supported Calling Conventions 42
IBM C Set/2 Calling Conventions 43
Borland C++ Calling Conventions 45
OS/2 Programming Models 46
About the Thunking Process 47
About Memory Allocation 48
Alternative Compiler Options 48
Calling Convention Options 48
Listing/Source Code Compiler Options 49
Using the NMAKE Utility 50
Resource Script Files 53
Module Definition Files 56

Chapter 4.
Tips on Printing in OS/2 2.1 59

Spooler Object 59
Spooler Space Requirements 59
Standard and Raw Data Spooling 61

Setting Print Priority 61
Printer Object 62
Pooling and Sharing 63
Print-Related Objects 64
Print Job Object 64
Previewing the Print Job Status 65
Previewing the Job Contents 66
Printer Driver Object 67
Printer Driver Extended Attributes 68
Creating a New Printer Object for a Printer Driver 69
Installing a New Printer Driver 70
Printer Drivers Provided with OS/2 2.1 71
Setting Printer and Job Properties 73
Deciding Which Printer and Job Properties to Use 74
Print Performance Options 76
Printer Port Object 77
Selecting an Output Port 78
Parallel Port and Serial Port Device Drivers 79
Queue Driver Object 80
Installing a Queue Driver 80
Setting Queue Options 80
OS/2 Print Utilities 81
Using Command Line Printing 81
Using Drag and Drop Printing 82
Using the Print Screen Key 82

Part II. Designing OS/2 2.1 Applications 85

Chapter 5.
Understanding Common User Access Interfaces 87

CUA and the Workplace Shell 87
Implementing the Object-Oriented Model 88
Applying Similar Behavior to Interface Controls 93
Presenting Consistent Visual Representations 95
Types of CUA Windows 95
Additional Visual CUA Components 97

Chapter 6.
Understanding the System Object Model 99

Defining Object Class Relationships 99
SOMObject 100
SOMClass 101

SOMClassMgr and SOMClassMgrObject 101
SOM Programming Features 102
Inheritance 102
Encapsulation 102
Polymorphism 103
SOM Run-Time Environment 103
Defining Object Classes 104
Creating a SOM Object Class 105
Creating a Class Definition File 105
Generating Files with the SOM Compiler 109
Building SOM Class Libraries 110
C File Template Stub Procedures 110
SOM Methods and Macros 111
Manipulating SOM IDs 112
Retrieving Object Information 113
Replacing SOM Functions 113
Debugging in SOM 113
Invoking SOM Methods 114
Method Invocation Pointers 114
Classifying an Object 115
Selecting the Appropriate Object Views 115
Determining the Appropriate Action Choices 115

Chapter 7.
Using the Workplace Shell Environment 117

Workplace Shell Programming Interface 117
Workplace Shell Classes 118
Modifying Pages in a Settings Notebook 120
Using Object Usage Methods 122
Using the Drag and Drop Feature 123
Using Setup Methods 124
Using Save/Restore State Methods 126
Understanding Pop-Up Menus 127
Working with Class-Specific Menu Items 128
Adding Items to Conditional Cascade Menus 128
Defining New Action Choices 129
Object Shadows 129
Using Open Views 129
Using a Details View 130
Obtaining Object Information 131
Defining Object Behavior 131
Object Templates 131
Using Objects in the Workplace Shell 132

Object Class Functions 133
Object Instance Functions 133
Creating Help for WPS Objects 134
Installing a Workplace Object 134
Running an Installation Program 135
Running REXX-Language Batch Files 135
WPS Class List Object Utility 136
Migrating Applications 136
Setting Up New Icons for Your Applications 139

Chapter 8.
Communicating among Machines Using TCP/IP 141

What Is TCP/IP? 141
Fundamental Protocols of TCP/IP 142
Other TCP/IP Protocols and Applications 143
Internet Addressing and Domains 143
Nameservers and Routers 145
Using TCP/IP in Your Application 145
Sockets 145
Remote Procedure Calls 146
Using External Data Representation 151
Less Common Protocols 151
Setting Up the Development Environment 152

Part III. Implementing Programming Solutions 153

Chapter 9.
Programming with the Modular Method 155

Using a Modular Approach to Coding 155
Using Function Modules 156
Types of Function Models 156
Using a Magic Window 157
Creating the Mainline Template 157
Window and Dialog Procedure Styles 160
Communicating with PM Windows 162
Sending a Message to a Window 163
Passing the Data as Parameters 164
Attaching Data to a Window 164
Creating a Make File 166
Additional Submodules for the Magic Window 168

Chapter 10.
Creating Effective Help for Your Application 177

Introduction to OS/2 Online Information 177
Online Information Features 177
Accessing the Online Tools 178
Using the IPF Compiler 179
Supporting International Languages 179
Concatenating Online Files 180
Creating a Standard Window 180
Understanding the Tag Language Syntax 183
  Userdoc Tag 183
  Text String Tags 183
  Using Tag Attributes 184
  Using Control Words and Special Characters 187
  Highlighting Online Text 187
  Creating Window Titles 188
  Determining the Contents Window Headings 189
  Creating Push Buttons 189
Indexing with IPF Tags 192
Creating Entries for the Master Help Index 193
Using the :Link. Tag 195
  Creating Hypertext Links 195
  Creating Hypergraphic Links 197
Creating Split Windows 197
  Creating Footnotes 198
Adding Help to an Application 199
  Setting Up the Help Tables 199
  Creating the HELPINIT Structure 201
Error Checklist 204
  Tagging Errors 204
  Indexing Errors 204
  Search Errors 205
  Hypertext Link Errors 205
  Hypergraphic Link Errors 205
  Display Errors 206
  Footnote Errors 206
  Miscellaneous Errors 206

Chapter 11.
The Warehouse Inventory Application 209

Warehouse Inventory Modules 210
  Procedure Modules 210

Function Modules 210
Function Collection Modules 211
Warehouse Inventory Main Module 211
Warehouse Inventory Main Window Procedure 216
Passing Data to a Dialog Procedure 216
Aligning Columns in a List Box 223
Dialog Procedures 225
Warehouse Inventory Submodules 229

Part IV. Reference Section 231

Chapter 12.
OS/2 Alphabetized Functions by Category 233

Dev Functions 233
Dos Functions 235
Gpi Functions 262
Prf Functions 307
Win Functions 308

Appendixes 359

Appendix A.
Reusable Submodules 361

Appendix B.
Header Files 373

Appendix C.
More Tips, Time-Savers, and Good Ideas 401

Design Tips 401
Setting Passwords and Screen Savers 402
Accelerator Keys and Shortcuts 402
Accelerator Keys 402
Other Shortcuts 403
Error Message Solutions 403
Remove Unnecessary Files and Programs 406

Glossary 409

Index 429

Preface

Note: Unless stated as new for OS/2 2.1, all references to the OS/2 operating system or OS/2 2.1 also pertain to OS/2 2.0.

This book is a valuable "quick start" reference for application programmers who use OS/2 2.1 as their primary operating system. By using the procedures we have provided, you will be able to immediately write and implement useful applications, even if you are a first-time user of OS/2 2.1.

Who Should Use This Book

This book is designed for programmers who have a basic knowledge of the C programming language and some exposure to the Presentation Manager or Windows *message-passing model.* In addition, you will need the IBM *OS/2 2.x Toolkit* and possibly the IBM *OS/2 2.0 Technical Library* and VNR *OS/2 Presentation Manager GPI.* We also recommend that you use the IBM *C Set/2* compiler for your compilations.

If you plan on using TCP/IP for OS/2 (see Chapter 8, "Communicating Between Machines Using TCP/IP"), you should have the IBM *TCP/IP Version 1.2 for OS/2, Base Package.* If you will be developing in TCP/IP for OS/2 2.1, you should also have the IBM *TCP/IP Version 1.2 for OS/2, Programmer's Toolkit,* which includes the IBM *TCP/IP Programmer's Reference.* We also recommend VNR's *Client Server Programming with OS/2 2.0.*

The examples included in this book are typical of, but not exclusive to, corporate usage. We have developed code modules, which are used in a *building block* format, to show you how to incorporate the most commonly used OS/2 functions to build major OS/2 applications with a minimum of experience.

In short, this book contains all the instruction and working code samples necessary to create efficient, "real-world" applications. Time that is usually lost in learning a new system is minimized by the many useful techniques, tips, and tricks of the trade provided in each chapter.

How This Book Is Organized

This book is organized into four parts and three appendixes:

Part I contains useful information on installing and using OS/2 2.1.

Part II contains an overview of the important design concepts and components of OS/2 2.1 that impact the development of a new application.

Part III contains a step-by-step process that demonstrates how OS/2 applications can be assembled from simple modules.

Part IV is a reference section that contains information on OS/2 functions, which are listed by category.

The **Appendixes** contain reusable submodules, header files, and additional tips and time-savers.

A complete glossary and index is provided at the back of this book.

Tips and recommendations based on our experience and research have also been included throughout the book. References to the books in the OS/2 2.0 Technical Library have been indicated, where appropriate, for more specific information.

Conventions Used in This Book

To provide a consistent format for easy reading and understanding, the following conventions are used throughout this book:

- New terms are *italicized* the first time they are defined. Functions, methods, and macros are italicized the first time they are introduced. Italics are also used for special emphasis of words or phrases.
- To avoid redundancy when more than one method of selection is available (for example, mouse, trackball, arrow keys, function keys, etc.), the mouse actions *click on* and *double-click on* have been used to denote the selection process.
- Menu options, checkboxes, and push buttons are usually written with the first letter capitalized (initial capitalization) and placed in boldface type. For example: **Settings** option, **HPFS** checkbox, **OK** push button.
- Except where otherwise noted, commands, file names, and extensions are capitalized, for example: SAVE command, CONFIG.SYS, .INI file.

- Displayed messages, prompts, and user instructions appear in special type. For example: GP Fault.
- Tip and Time-Saver boxes are shaded and placed in relevant sections of each chapter.

About the Companion Diskette

A companion diskette for this Handbook is included. This diskette contains all the code and functions used to implement the modular applications described in this book. You can save many hours of work by incorporating these written and tested code modules into your application design.

Special Thanks

The following people have been invaluable in assisting us with the organization and content of this book:

- Chris Andrews
- Craig Bennett
- Ivan Bittles
- Michael W. Brown
- Richard Bucker
- Marilyn Dichtenberg
- Dan Lee
- Michael O. Lee
- Marion Lindsey
- Marcelo Lopez
- Peter Magid
- Scott Nesbitt
- Michael Perks
- Noriko Rajogopolan
- Frank Schroeder
- Uri Stern
- Vickie Szarek
- Shixiong Yang, PhD.

Those of us who have participated in the development of OS/2 2.1 are extremely pleased with this product because of its overall look and feel, ease of use, and scope of compatibility. We think you will be, too.

Part I. Tips on Installing and Using OS/2 2.1

Chapter 1.
Key Features of OS/2 2.1

OS/2 2.1 is considered by many to be the operating system of the future. You no longer need to choose between OS/2, DOS, or Windows because OS/2 2.1 *contains* DOS and Windows. This powerful operating system enables you to retain your old program libraries while adding new OS/2 applications.

Some of the key features of OS/2 2.1 include preemptive multitasking, task scheduling, 32-bit addressing, Boot Manager (a multiboot option), virtual memory for use with multiple DOS sessions, and fast, hard-disk file access using the High Performance File System (HPFS). See Chapter 2, "Tips on Installing OS/2 2.1" for more information on Boot Manager and HPFS.

Other new or improved features of OS/2 2.1 are described below.

CD-ROM Installation

OS/2 2.1 can now be installed from CD-ROM. This process takes less time and is less complicated than the ordinary installation from diskette.

Improved and Additional OS/2 2.1 Support

Support for Online Linking and Embedding (OLE), PCMCIA bus, and Advanced Power Management (APM) has been improved or included in OS/2 2.1.

Additional Device Driver Support

The Selective Install program now provides support for XGA display drivers, SVGA display drivers, SCSI adapters, and the initial printer installation. Device driver (OS2CDROM.DMD) and file system (CDFS.IFS) support is also provided for both IBM and non-IBM CD-ROM drives.

Windows 3.1 Support

WIN-OS/2 now supports Windows 3.1 applications running in standard mode and some Windows 3.1 applications running in enhanced mode. OS/2 2.1 also provides a set of Windows 3.1 printer drivers that can be used with WIN-OS/2.

MMPM/2 Support

Multimedia Presentation Manager/2 (MMPM/2) is now included with OS/2 2.1 and provides support for OS/2 PM multimedia applications.

Dual Thread Support

In OS/2 2.0, DOS applications use separate *threads of execution* to perform their operations; the number of available threads depend on the makeup and complexity of the system and the other running applications. In OS/2 2.1, support is provided for *dual threads* to enable one thread to perform read/write operations while the other thread services any interrupts. (To enable the dual-thread feature, the DOS Setting INIT_DURING_IO must be set to ON.)

Fewer Performance and Memory Restrictions

Many restrictions caused by the performance and memory limitations of DOS and Windows are no longer a problem under OS/2 2.1. Multiple DOS, Windows, and OS/2 applications can run simultaneously in the OS/2 operating system. DOS applications now run better than ever under OS/2 2.1 because they have more internal memory available to them and Windows applications run better under OS/2 2.1 than they do under DOS! In addition, DOS and OS/2 applications can now be started from a Windows application. This is useful for any Windows applications that depend on DOS utilities.

32-Bit Flat Memory Model

OS/2 2.1 uses the 32-bit *flat memory model* of the Intel 80386 microprocessor, which allows linear addressing of memory to be fully implemented. The flat (non-segmented) memory model provides a higher performance level in the OS/2 2.x versions than in previous versions, which used the 16-bit *segment:offset* model. Other benefits include a single memory object for applications, better memory-usage tracking,

protected-mode memory, and 32-bit calling conventions that reduce execution time by eliminating segment loads.

32-Bit Graphical Environment

You can take full advantage of the 386, 486, and upcoming Pentium microprocessors and write true 32-bit applications using the OS/2 32-bit graphical environment. OS/2 2.1 can run 16- and 32-bit applications simultaneously due to *tiled addressing* and a mechanism called *thunking*, which enables 16-bit segmented architecture to interface with the 32-bit, flat memory model. The effect is transparent to the user. See Chapter 3, "Tips on Compiling Your OS/2 Application" for more information on thunking.

Multiple Virtual DOS Machine

A significant feature in OS/2 2.1 is the *Multiple Virtual DOS Machine (MVDM)*, which enables each real-mode DOS task to act as if it were performing in its own environment. By using the virtual 8086 mode of the 80386 microprocessor, each DOS application is protected from other applications running in the system. If one of your applications fails, only that application is terminated; all other programs will continue to run and the operating system will not remain intact.

Each MVDM provides 630KB of free, base memory. MVDM also supports the LOTUS/INTEL/MICROSOFT (LIM) Expanded Memory Specification (EMS) 4.0 emulation, the Extended Memory Specification (XMS) 2.1, and the DOS Protected Mode Interface (DPMI) Version .9+.

EMS and XMS can be specified per VDM using the DOS Settings, EMS_MESSAGE_LIMIT and VDM_MESSAGE_LIMIT. DOS Settings are located on the Sessions page of the VDM .

Workplace Shell

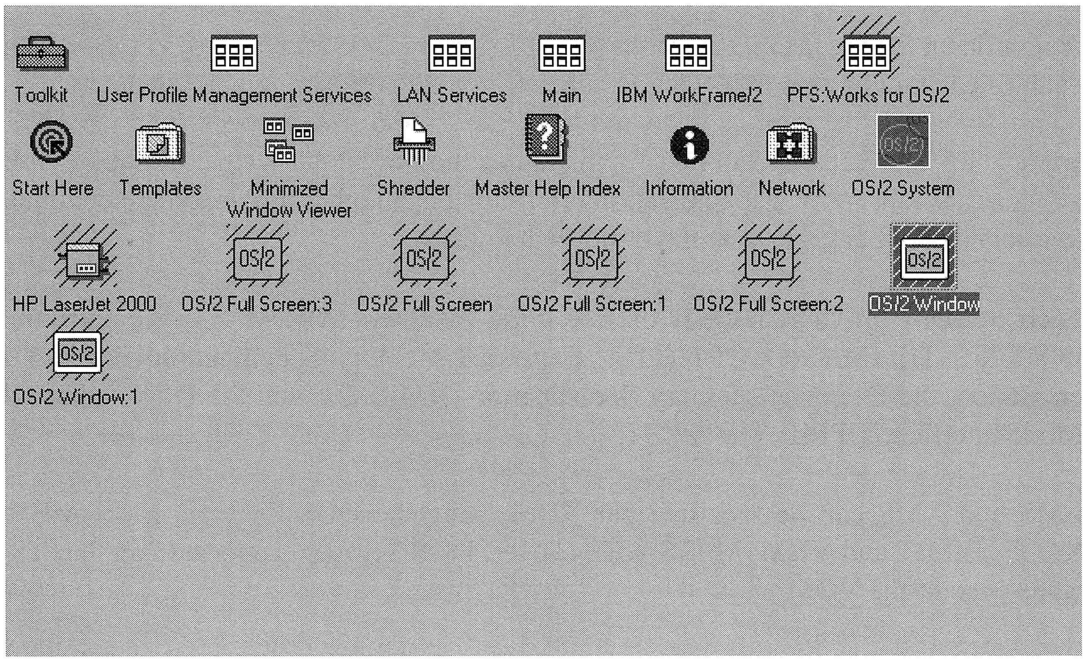
The OS/2 *Workplace Shell (WPS)* combined with the OS/2 *Presentation Manager (PM)* allows new users to quickly start their applications. Based upon the 1991 CUA guidelines, this object-oriented environment enables the user to directly interact with an object, replacing the need to type commands on the command line. See Chapter 5,

"Understanding Common User Access Interfaces," Chapter 6, "Understanding the System Object Model," and Chapter 7, "Using the Workplace Shell Environment."

OS/2 Desktop

The OS/2 Desktop is a screen arranged to look and work like an ordinary desktop (see Figure 1-1). The Desktop items are represented by *icons*, which are pictorial representations of real objects. By double-clicking the left mouse button on one of these icons, you can start an application, perform a function, or open a folder. (A *folder*, which consists of a window frame and a container control object, contains groupings of applications, objects, or functions that are arranged by categories.) Icons with hatch marks represent started applications.

Figure 1-1. Example of an OS/2 2.1 Desktop



Command Prompts and System Folders

For those who feel more comfortable starting applications by using the command prompt to type commands into the system, OS/2 2.1 provides a Command Prompts folder, which contains objects that provide access to the DOS and OS/2 command prompts.

OS/2 2.1 also provides an OS/2 System folder that contains a group of subfolders. One of these subfolders, called the Productivity folder, contains many new useful programs that allow users to manage their time, plan activities, use a spreadsheet, etc. Another subfolder, containing games such as Solitaire, Chess, and Reversi, is available just for fun.

Settings Notebook

The Workplace Shell has a control called *Notebook*, which enables a multipage dialog to be presented to the user in a book-like form. A *Settings notebook* contains the settings for a WPS object. The settings are grouped together in a notebook format: *tabs* indicate the type of setting and *pages* list the available options for that tab. The Settings notebook is useful for complex objects because it provides familiar representations to the user instead of a series of windows with menus and submenus. See Chapter 7, "Using the Workplace Shell Environment" for more information on Settings notebooks.

Clipboard Interactivity

The *clipboard* feature enables users to cut or copy information from one window and *paste* it into another window. For example, results from a spreadsheet can be copied (clipped and pasted) into a word processing program; and a Window application's screen output can be pasted into an OS/2 window. This interactivity between the individually running processes has been greatly improved in OS/2 2.1, especially in data exchanges between WIN-OS/2 and PM applications.

Integrated LAN Resources

Access to local area networks (LANs) has been greatly improved through the use of the *LAN independent shell*. This shell is supported by several network protocols, including Novell Netware and the IBM LAN Server. Previously, a user had to switch to the full-screen interface of the LAN Requester to use LAN resources. Because the LAN independent shell has been integrated into the Workplace Shell, LAN resources can now be moved into a folder on the OS/2 Desktop for easier access. In addition, a network printer can be defined as the default printer, freeing the local printer for more specific jobs.

Improved Online Help

The comprehensive online help provided with OS/2 2.1 assists new users in understanding the use of the mouse, icons, and other unfamiliar terms and features. See Chapter 10, "Creating Effective Help for Your Application."

Chapter 2.
Tips on Installing OS/2 2.1

If you have successfully installed OS/2 2.1 and it is running to your satisfaction, you can skip this chapter. However, the tips provided here might give you a smoother running operating system.

The installation process for the OS/2 2.x versions has been greatly enhanced over previous versions of OS/2 because it uses a graphical user interface that has full mouse and keyboard support. For the average user, the installation should cause no problems and take about 45 minutes when loading by diskette. However, depending on the options selected and the processing speed of your computer, it might take longer.

OS/2 2.1 can be installed from a local area network (LAN). Under a LAN installation, optional features can be specified globally by a network administrator or specifically by an end user. This method of installation takes about 20 minutes.

OS/2 2.1 can be installed from a CD-ROM. This method also takes about 20 minutes.

In this chapter, we have focused on the most common means of installation (diskette) and the most common problem of setting up partitions for Boot Manager to clarify which options might be most useful to you.

The recommended installation sequence is as follows:

1. Base OS/2 2.1
2. IBM WorkFrame/2 (optional)
3. OS/2 2.x Toolkit
4. IBM C Set/2 (optional)
5. Base TCP/IP (optional)

Before Installation: Back Up Your Applications

If you have been running DOS and Windows applications, back up your applications before installing OS/2 2.1. This will ensure that your files are preserved if you decide to format your hard disk or partition during installation. You might also want to reboot

your system to avoid potential initialization problems that could occur because of the previous (*leftover*) hardware state.

What To Do First

Take some time to consider your system needs -- a little planning at this point can save many problems later. If this is your first OS/2 installation, start by asking yourself the following questions:

Do I need to boot from more than one operating system?

Most applications work well when running under OS/2 2.1 and make the need for more than one operating system unnecessary. If you decide you need separate DOS, Windows, and OS/2 operating systems, you should use the multiboot option (see "Installing Boot Manager " on page 13).

How should I set up my drives?

OS/2 2.1 is a very powerful operating system that must be set up properly to allow it to function efficiently. Partition size, placement, and setup are important. (We recommend a minimum of 50MB for an OS/2 partition.) See the partitioning example on page 19 and "Disk Drive Considerations" on page 22.

Do I have to reinstall all my DOS and Windows applications in order to use them under OS/2 2.1?

Reinstallation is not usually necessary because OS/2 2.1 enables you to migrate applications during installation or from the Desktop. See Chapter 7, "Using the Workplace Shell Environment" for information on how to migrate your applications through the Workplace Shell.

However, if you have installed OS/2 on Drive C and your DOS and Windows applications are on Drive D, or if you have backed up your applications onto tape or diskette, you may need to reinstall your applications. (Some applications may need to be reinstalled even if these conditions are not present; determining which ones to reinstall is often a matter of trial and error.)

Do I have to format my hard drive in order to install OS/2 2.1?

No. OS/2 2.1 will replace the operating files of any previous version of OS/2 but not your programs or data. However, formatting your hard drive is the safest way to ensure

a clean installation. (If you intend to set up HPFS, you *must* format your hard disk. See "Installing the High Performance File System" on page 21.)

Can I install multimedia support for audio adapters such as Sound Blaster?

Yes. After the OS/2 2.1 installation is completed, install the Multimedia Presentation Manager/2, which is part of OS/2 2.1. MMPM/2 provides support for audio adapters, system sounds, playing music CDs, and viewing high-resolution digital movies.

If I have installed unsupported hardware or wish to install new hardware, do I have to reinstall OS/2 2.1?

In most cases, you can change your configuration without having to reinstall OS/2 2.1. Simply double-click on the OS/2 System icon on the Desktop, double-click on the System Setup icon, and choose **Selective Install**. This option will display the System Configuration screen and allow you to make changes to the default options. However, if your selections prevent you from bringing up OS/2 2.1, reinstallation may be necessary. See "Installing OS/2 2.1 from Diskette: Step 1" on page 13.

What kind of devices does OS/2 2.1 support?

OS/2 2.1 supports many different types of devices such as CD-ROM, SCSI, and SVGA. However, not all combinations of adapters and devices are supported or fully supported. For example, OS/2 2.1 provides only basic support for Super VGA (SVGA) devices. Consult your OS/2 2.1 documentation and your device or adapter documentation for support information and specific installation instructions.

Microprocessor Requirements

OS/2 2.1 has been designed to exploit the features of the Intel 80386 microprocessor such as the 32-bit flat memory model, the virtual 8086 mode (which provides for the Multiple Virtual DOS Machine (MVDM)), and the ability to address up to 4GB of memory in a single logical unit. Therefore, OS/2 2.1 only runs on 80386SX or higher processors. However, OS/2 2.1 will work on other manufacturer's processors besides Intel if they are designed to emulate the 80386.

System Memory Requirements

Because of the many capabilities of OS/2 2.1, sufficient memory is required to enable the system to perform quickly and efficiently. The minimum amount of memory that

OS/2 2.1 requires is 4MB. However, more memory increases performance because the more functions the operating system can perform in memory, the less it has to use the swapper file.

In a minimum (low) memory system, requests for memory can exceed supply, causing a *memory contention*. This often occurs when starting too many applications. (To avoid this situation, terminate the first application before starting the second.)

Whenever the system has a memory contention, memory pages that are not immediately required are swapped out to a disk file called SWAPPER.DAT. This process, known as *paging out*, is much slower than memory paging because the least recently used pages (4KB blocks) of memory must be moved to disk so that they can be freed up to be used by another application.

During complex operations, a system with low memory will sometimes appear to halt. Unless you have immense patience, we strongly recommend you load 8MB or more of memory on your computer.

TIP

When adding memory, make sure that the added memory chips are of the same speed and, if possible, the same manufacturer as the current ones or memory errors (Trap 2) might occur.

Hard Disk Space Requirements

OS/2 2.1 requires approximately 36-40MB of hard disk space. Depending on how many options you install, the system will use up to 33-36MB of disk space. Approximately 3-4MB are used for the swapper file, which can grow in size depending upon the system memory available and the number of processes that are running.

Unless you have a lot of space on your hard disk, plan on moving the swapper file during or after installation to your last logical drive (for example, *E:*\). See "Disk Drive Considerations" on page 22 for directions on how to move the swapper file.

Installing OS/2 2.1 from Diskette: Step 1

The installation procedure from diskette consists of two steps: installation and configuration. The first step involves the Installation Diskette and Diskettes 1 through 5. This step loads the basic operating system that supports the graphical installation program and allows you to install Boot Manager.

If you plan to use more than one operating system and want to take advantage of the Boot Manager menu, it is important that the partitions for your hard drives be set up correctly. Because this can be a rather complex process, we have provided a description of the Boot Manager installation procedure with suggestions on how to partition your hard disk.

Installing Boot Manager

Boot Manager is a feature of OS/2 2.1 that enables you to use multiple operating systems on one machine. For example, it is possible to have DOS, previous OS/2 versions, and OS/2 2.1 all installed on the same computer. In addition, Boot Manager enables OS/2 2.1 to be installed and booted from a logical disk other than Drive C. However, it requires sufficient disk space (equal to the total disk space needed by all the operating systems being installed plus 1MB for Boot Manager) to support multiple operating systems.

When you install Boot Manager, you must separate your available hard disk space into *partitions*. These partitions are assigned to the operating systems (for example, DOS and OS/2 2.1) that are to be set up. Each disk can be divided into several *primary partitions* and/or one *extended partition*. (The extended partition can replace one of the primary partitions. It is created during the creation of your first logical drive and will contain all the logical drives that you create.)

Boot Manager supports up to four primary partitions (including itself) per fixed disk, however, only one primary partition is accessible at any given time. (For example, when running OS/2 from a primary partition, you cannot access DOS on another primary partition. However, you can access files on a logical drive or an extended partition.) The first physical disk drive in the system *must* have a primary partition.

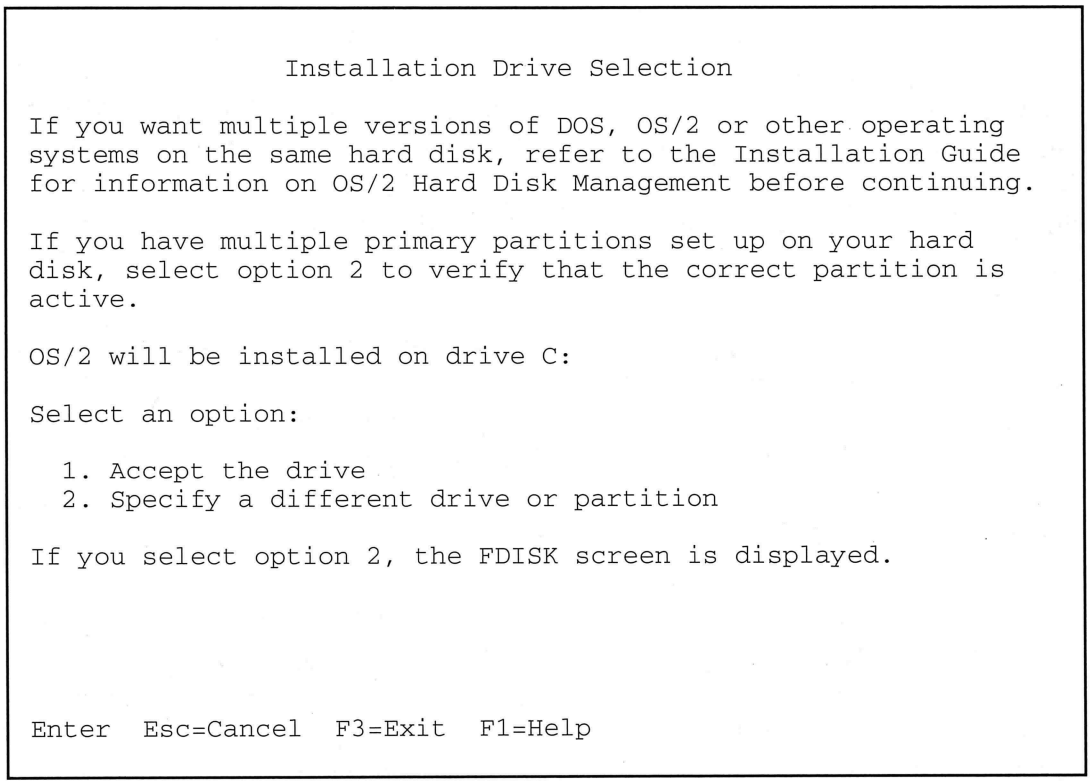
During the OS/2 installation, the Installation Drive Selection screen will be displayed (see Figure 2-1 on page 14). You will be asked for the drive on which you wish to

install OS/2 2.1. If you had a previous version of OS/2 on your system and your partitions are already set up correctly, you can select option 1 (**Accept the drive**). However, you may want to format your partitions later in the installation process to ensure that you have a clean install with no leftover files from the previous OS/2 version.

To install Boot Manager, select option 2 (**Specify a different drive or partition**). This will display the FDISK screen, which is used to manage partition information on your fixed disk.

Make sure that there is some free space available on which to install Boot Manager. (If necessary, you can delete a partition through the **Options** menu. However, deleting a partition will cause all data stored on that partition to be lost.)

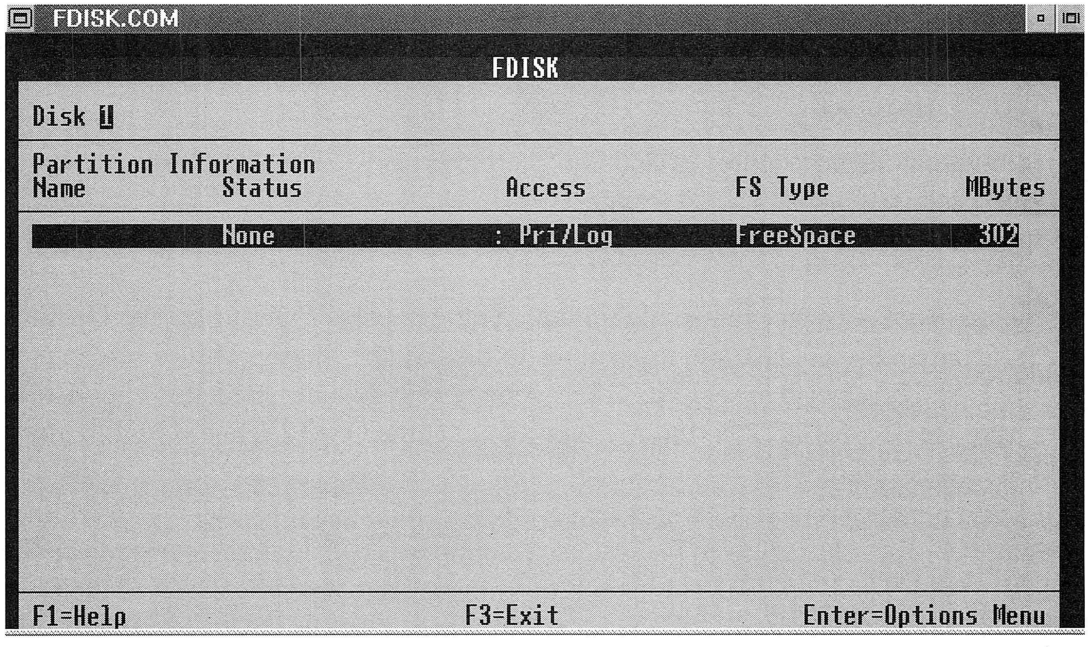
Figure 2-1. Installation Drive Selection Screen



A Warning screen will appear reminding you to back up all necessary data before proceeding. If your files are backed up, press Enter to continue.

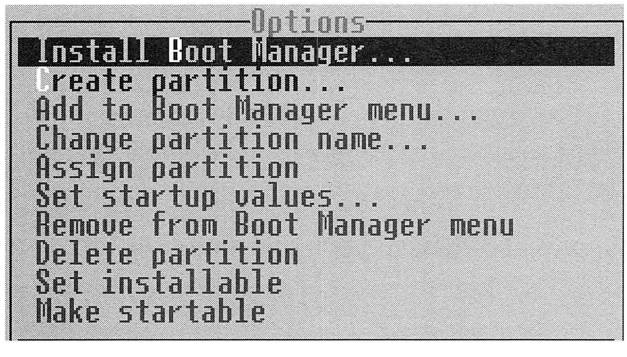
Once you have a *Free Space* line on the FDISK screen, make sure that it is highlighted (see Figure 2-2).

Figure 2-2. FDISK Screen



Then press Enter to display the **Options** menu (see Figure 2-3), select **Install Boot Manager**, and press Enter to continue.

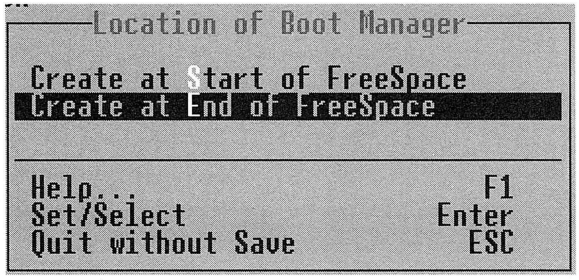
Figure 2-3. FDISK Options Menu



This will display the Location of Boot Manager pop-up window (see Figure 2-4 on page 16). Make your selection (we recommend that you place Boot Manager at the

end of your partitions to avoid storage fragmentation) and then press Enter to return to the FDISK screen.

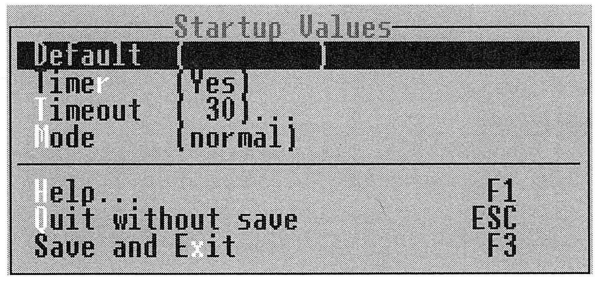
Figure 2-4. Location of Boot Manager Pop-Up



While the Boot Manager line is highlighted, you can press Enter to get the **Options** menu again and select **Set startup values** to display the Startup Values screen (see Figure 2-5). If you wish to set OS/2 2.1 as your system default, select this option after you have created the OS/2 partition. Make sure that the OS/2 2.1 line is highlighted.

The Startup Values screen allows you to set the Boot Manager *timer*, which determines the number of seconds allowed for system selection before Boot Manager starts up the default (highlighted) operating system. If the timer is set to zero (*0*), however, the Boot Manager Menu (see Figure 2-11 on page 20) will not be displayed and the default system will boot immediately.

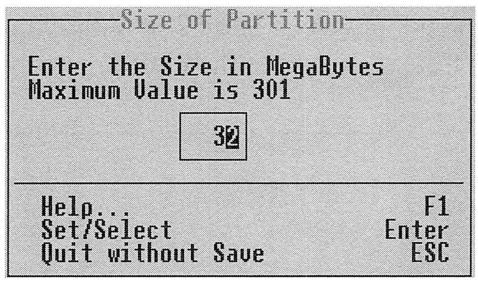
Figure 2-5. Startup Values Screen



After you have made your selections, press F3 to save your selections and return to the FDISK screen.

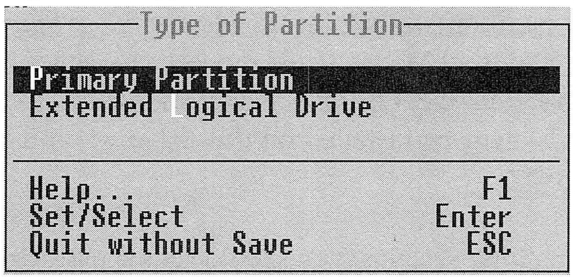
To create your first partition, highlight the Free Space line and press Enter to go back to the **Options** menu. Select **Create partition** to display the Size of Partition pop-up window (see Figure 2-6).

Figure 2-6. Size of Partition Pop-Up



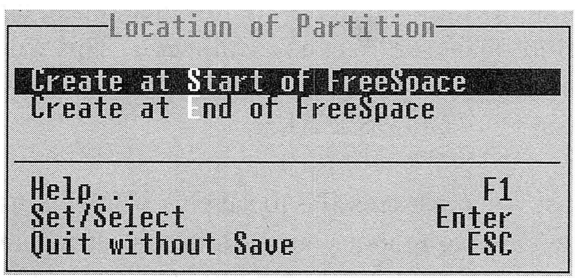
If you plan to use DOS as one of your operating systems, it must be in a primary partition and should be the first system installed on Boot Manager because DOS must be initiated from Drive C. Set the size of this (in this example, DOS) partition, and then press Enter. A Type of Partition pop-up will appear (see Figure 2-7).

Figure 2-7. Type of Partition Pop-Up



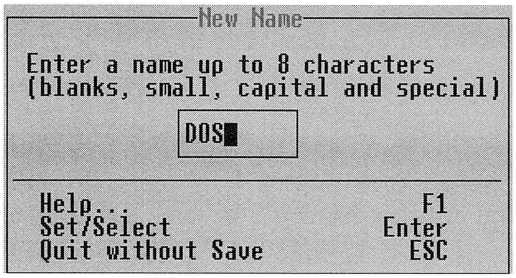
If this is for DOS, you should make this a primary partition for Drive C. (You must create a least one primary partition for the first hard disk in your system.) Press Enter to display the Location of Partition pop-up (see Figure 2-8). Make your selection and then press Enter to return to the FDISK screen.

Figure 2-8. Location of Partition Pop-Up



To add this partition to Boot Manager, go to the **Options** menu again and select **Add to Boot Manager menu** to display the New Name pop-up (see Figure 2-9). After you have entered the name (in this case, DOS), press Enter to return to the FDISK screen.

Figure 2-9. New Name Pop-Up



To place OS/2 2.1 on Drive D, set up the size of your next partition, set the partition type (we recommend that you set the type to *Extended Logical Drive* because you cannot access data from another primary partition), set the location to *Create at Start of Free Space*, which will add the partition to Boot Manager and request a name for it such as OS/2 2.1. At this point, go back to the **Options** menu and select **Set installable**. This will ensure that OS/2 2.1 will be installed on this drive when the system is restarted and you accept the drive.

To partition additional DOS and OS/2 operating systems, you should set the partition size for each system you wish to add, set the type of partition to *Extended Logical Drive*, set the location to *Create at Start of Free Space*, and then add the partition to Boot Manager.

Only one partition can be set to installable. To begin your next installation, you will need to restart OS/2 2.1, bring up the FDISK screen, and set the next partition you want to install as *Installable*. After all your operating systems are installed, bring up the FDISK screen again to confirm that Boot Manager is set up as *Startable*. This will ensure that the Boot Manager Menu is displayed. Be sure not to install this new operating system on the drives that contain DOS and OS/2 2.1.

When you have created all your partitions, you must press F3 to save your selections and exit the FDISK screen. You will now need to reboot your system to ensure that your partitions are recognized and to continue the OS/2 installation. After rebooting, the Install Drive Selection screen will display again, informing you that OS/2 will now be installed on Drive D. Select option 1 to accept the drive and continue the OS/2 installation process.

WARNING

If you install a separate DOS operating system under Boot Manager, it *must* be initiated from, and therefore installed on, Drive C.

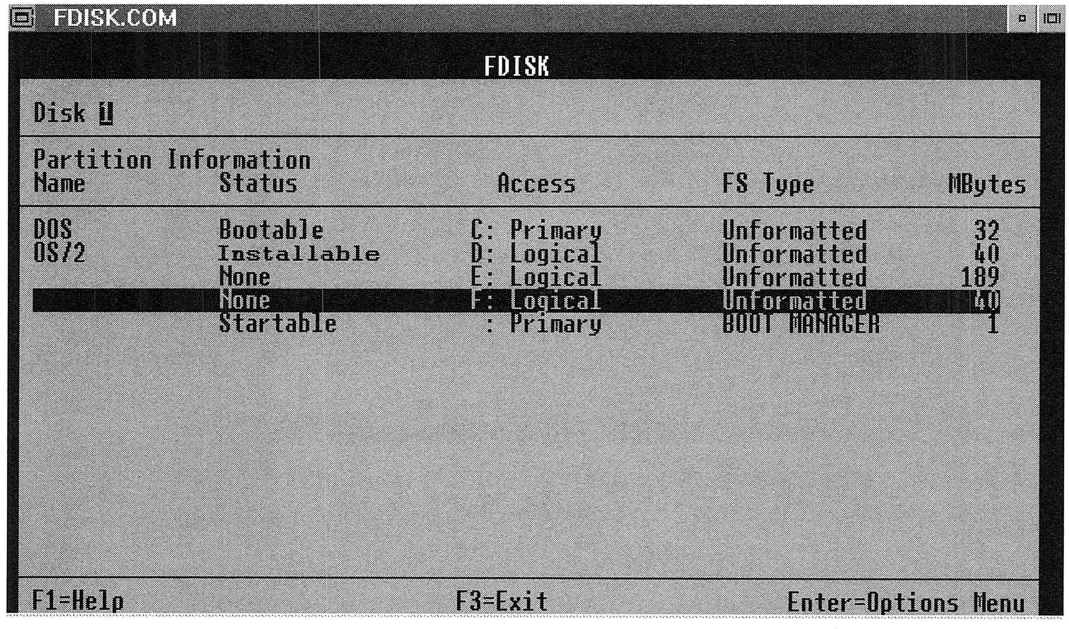
If you install DOS on a partition *after* installing OS/2 2.1 on that same partition, DOS will overwrite the OS/2 operating system.

If you install DOS 5.0 on Drive C after installing OS/2 2.1 on Drive D, Boot Manager will become inoperable. If this occurs, you should type FDISK at the DOS command prompt and then select Option 2 to set the active partition. When you are asked where to place the primary partition, select the non-DOS, 1% partition. This is the Boot Manager partition. Press F3 to save your changes and reboot your system.

Example of Partitioning and Boot Manager

Figure 2-10 shows an example of a 302MB hard drive that is set up with partitions and Boot Manager.

Figure 2-10. Partitioning Example

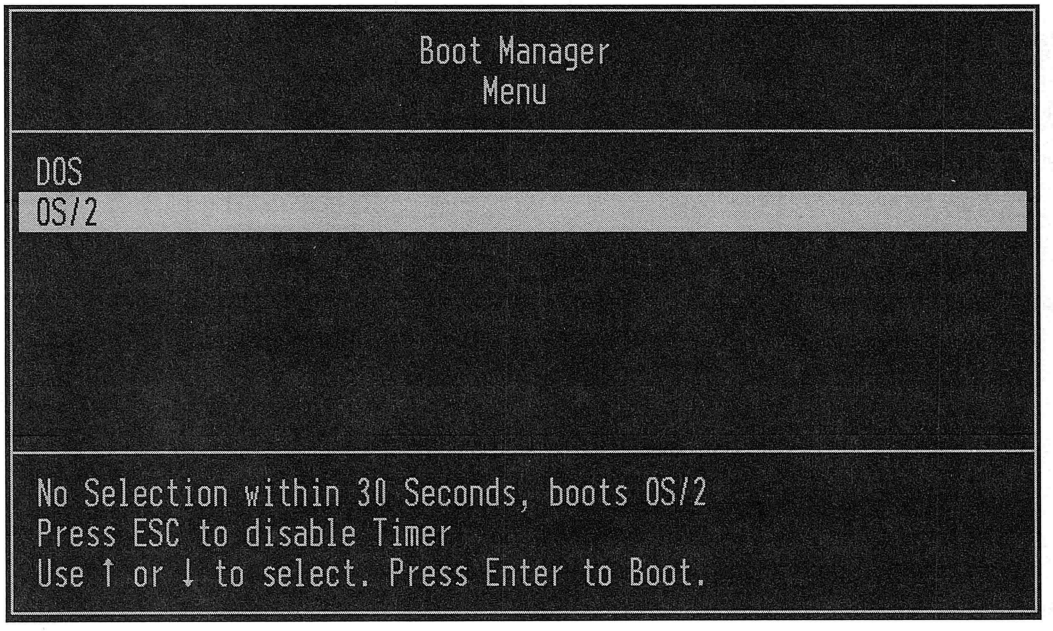


The partitions for this hard disk are set up as follows:

- The DOS partition has a Status: *Bootable*, Access: *Primary*, FS (File System) Type: *Unformatted* (DOS can only support FAT), and MBytes: *32*. It is assigned to Drive *C*.
- The OS/2 partition has a Status: *Installable*, Access: *Logical*, FS Type: *Unformatted*, and MBytes: *40*. It is assigned to Drive *D*.
- The third partition has a Status: *None*, Access: *Logical*, FS Type: *Unformatted*, and MBytes: *189*. It is assigned to Drive *E*.
- The fourth partition has a Status: *None*, Access: *Logical*, FS Type: *Unformatted, MBytes: 40*. It is assigned to Drive *F*. (Because this is the last logical drive, the swapper and spooler files should be moved here.)
- The Boot Manager partition has a Status: *Startable,* Access: *Primary*, FS Type: *BOOT MANAGER*, and MBytes: *1*.

When your computer is powered up, the Boot Manager Menu (see Figure 2-11) will appear and allow you to select an operating system. If you do not make a selection and press Enter within the number of seconds set by the timer, the operating system that was highlighted when the Boot Manager Menu was first displayed will be booted.

Figure 2-11. Boot Manager Menu



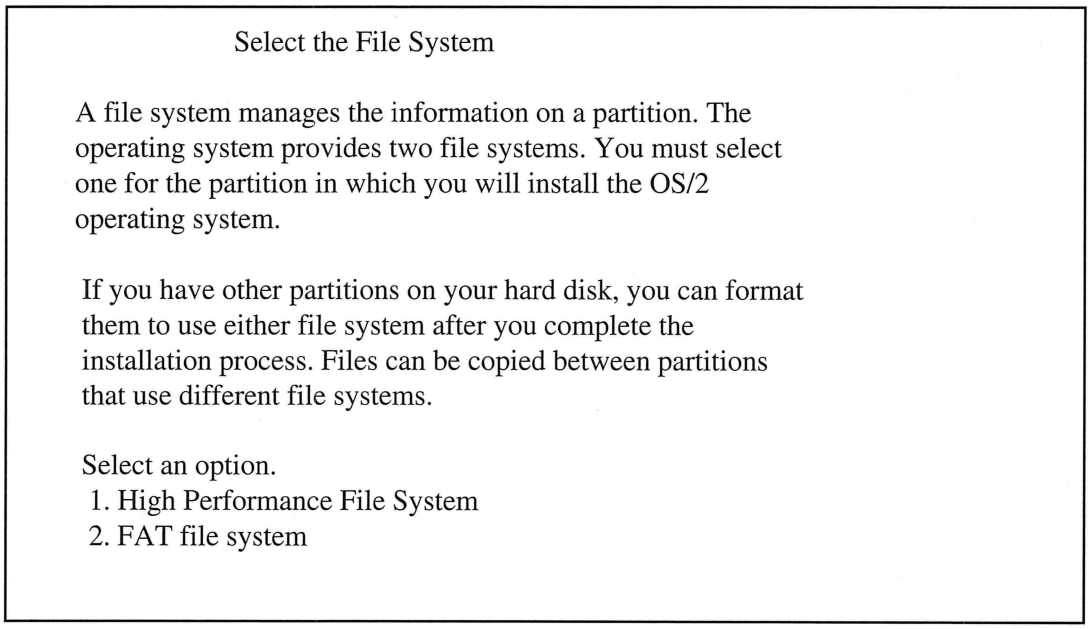
You can change the timer settings at any time by typing *FDISKPM* on the OS/2 command line to bring up the Fixed Disk Utility screen. Make sure the *Boot Manager* partition line is highlighted on the screen, then click on the **Options** menu and select **Set startup values**. This will display the Startup Values screen where you can change the timer settings. After making your changes, click on the **Set** button to return to the Fixed Disk Utility screen. Be sure to press F3 to save your changes.

Installing the High Performance File System

OS/2 2.1 has an option called the *High Performance File System (HPFS)*, which is faster than the *File Allocation Table (FAT)*. HPFS allows up to 254 mixed-case characters and imbedded blanks in file names. These file names are easier to identify than the 12-character file names allowed by FAT.

To install HPFS for OS/2 2.1 on your installable drive (in this case, Drive D), select the **High Performance File System** option on the Select the File System screen (see Figure 2-12).

Figure 2-12. Select the File System Screen



The DOS operating system cannot recognize HPFS-formatted partitions. HPFS files can be detected only by DOS or WIN-OS/2 sessions running under OS/2. For

example, assume that three partitions have been set up as: DOS (Drive C, FAT), OS/2 (Drive D, HPFS), and a logical drive (Drive E, FAT).

If you boot under DOS instead of OS/2 2.1, the HPFS files will be ignored and Drive E will be referred to as Drive D. Because this can cause considerable confusion, we recommend that HPFS be installed only when it is on the *last* logical drive, or when OS/2 2.1 is used as the only operating system.

Disk Drive Considerations

Even if Boot Manager is not installed, it still is wise to set up your system as three logical drives. The first logical drive, Drive C, should be at least 36-40MB because the initial installation sets up a 3-4MB swapper size, and 33-36MB is the minimum used if all the install options are selected. If you have elected to do a minimum install (fewer options), this drive size can be reduced to about 18MB.

TIP

If you are using Drive C for OS/2 2.1, we recommend you use your second drive (Drive D) for your applications. When you install future versions of OS/2 on your first drive (Drive C), you won't have to reinstall your applications if you have to format the C partition.

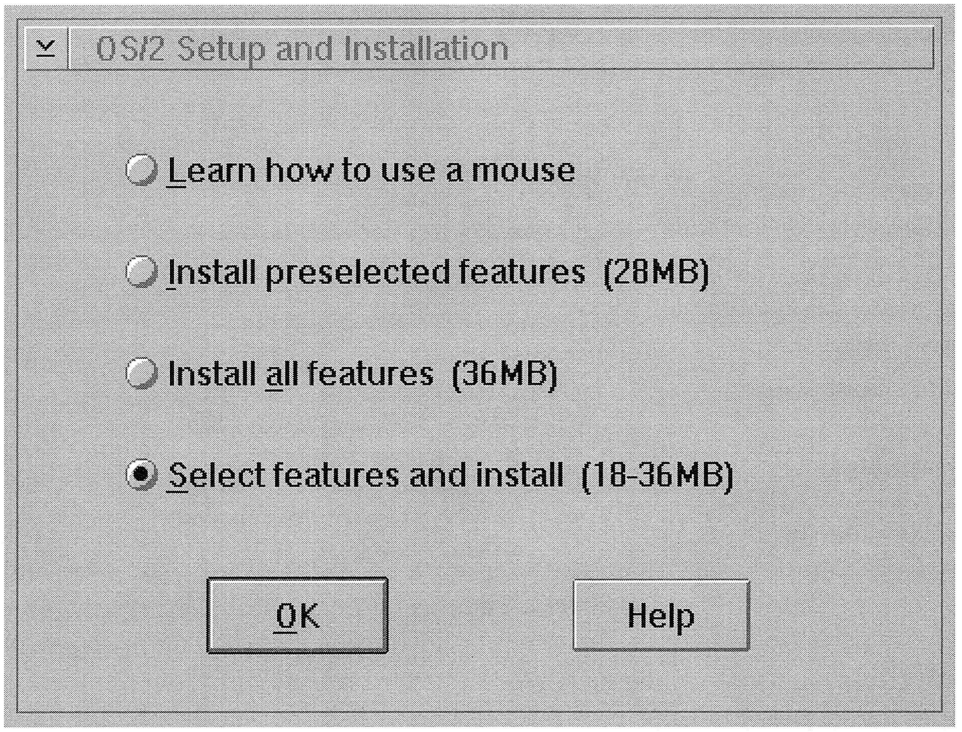
It is a good idea to move the swapper and spooler files from Drive C to the last logical drive, which (depending on memory availability) should be approximately 40MB to allow for adequate growth. This will enable you to add system-related file options such as additional printers or bit maps. See "Moving the Swapper File" on page 26.

Installing OS/2 2.1 from Diskette: Step 2

The second step in installing OS/2 2.1 enables you to configure your system. After your partitions have been set up and you have rebooted your system, the OS/2 Setup and Installation screen (see Figure 2-13) is displayed.

The two most frequently selected options on this screen are **Install all features** and **Select features and install**.

Figure 2-13. OS/2 Setup and Installation Screen

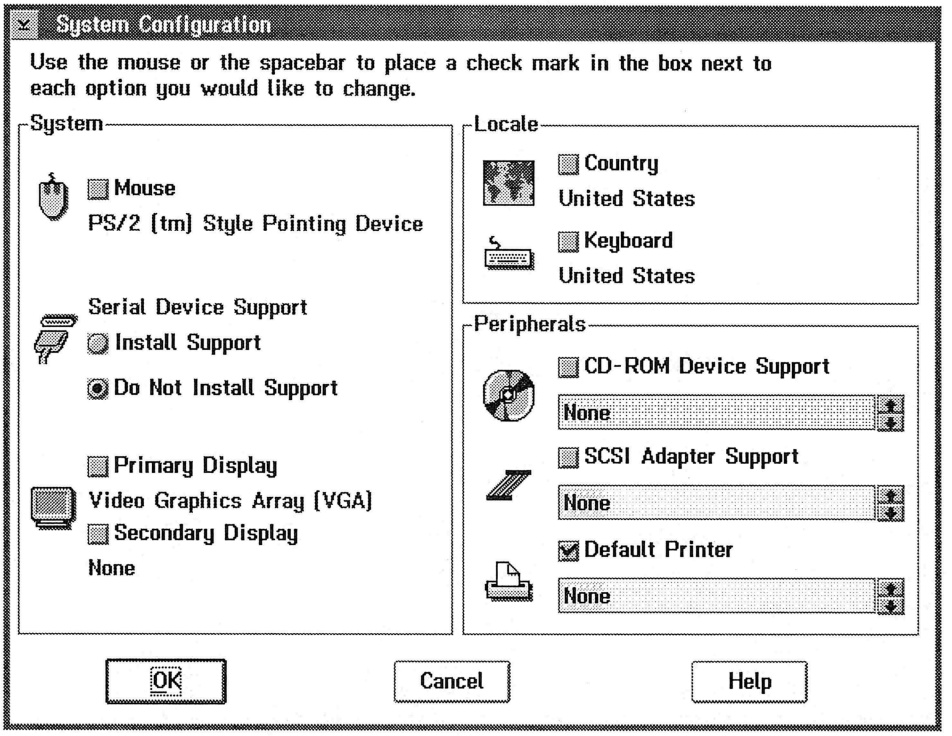


If you have enough disk space as shown in parentheses on the screen (in this example, 36MB or more), you can install all of the OS/2 2.1 features by selecting **Install all features**. Note that these numbers will vary, depending on your system and the available memory.

Installing only the features you need or want will save disk space. To do this, choose the **Select features and install** option. The System Configuration screen will be displayed (see Figure 2-14 on page 24). A list of available options are displayed on this screen with the default settings determined by your system hardware.

If the configuration later needs to be modified, you can change it using the **Selective Install** option in the System Setup folder (in the OS/2 System folder located on the Desktop), unless you have selected hardware options that prevent the startup of OS/2 2.1. In this case, you may have to reinstall the operating system.

Figure 2-14. System Configuration Screen

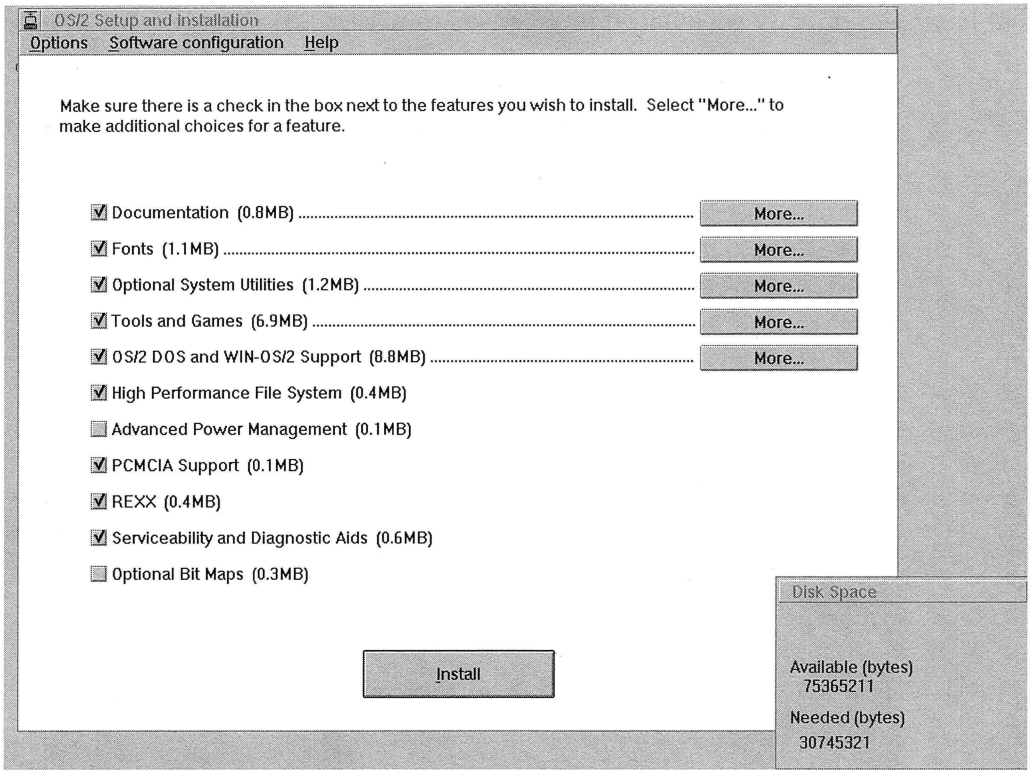


After making your selections, click on **OK**. The Select Printer(s) screen will appear. Make your default printer selection and click on **OK** to return you to the System Configuration screen. (You can select only one printer during the OS/2 2.1 installation. Use the **Selective install** option from the System Setup folder to select additional printers after installation.) Click on **OK** again to display a second OS/2 Setup and Installation screen where you can indicate which optional features you wish to install.

OS/2 Setup and Installation Screen

This screen appears after the System Configuration screen and lists the optional installable features that are available. Each option displays the feature's name and the amount of hard disk space required to install it. To select or deselect an item, click on the checkbox. Click on the **More...** push button (if displayed for that option) to further select or deselect options. The bytes available on your hard disk and the bytes needed to install the selected options will appear in the lower-right corner of the screen (see Figure 2-15).

Figure 2-15. OS/2 Setup and Installation (2nd) Screen



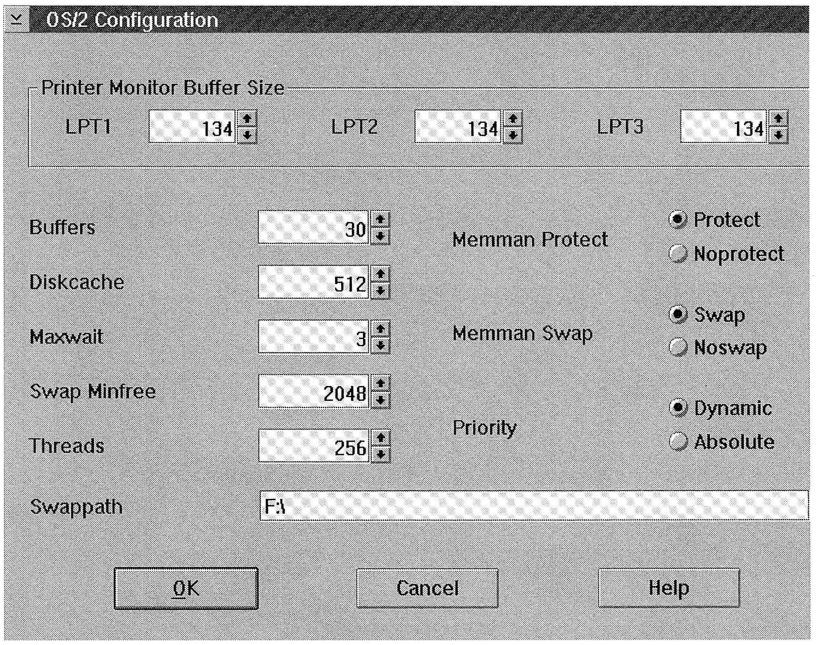
At this point, you can choose to format your partitions by selecting **Options** from the menu bar, clicking on **Format**, and indicating which file system you want to use for each of your partitions.

You can continue with the installation process or continue to customize your installation. To continue the installation process, click on the **Install** button. The operating system will prompt you to load the remaining diskettes. During the installation, a graphical progress indicator will show which files are being loaded from each diskette.

To further customize your installation, click on the **Software configuration** menu option located on the menu bar at the top of the screen *before* clicking on the **Install** button. This will bring up a pop-up menu where you can select either **Change OS/2 parameters** or **Change DOS parameters**. Depending on your selection, the OS/2 Configuration screen (see Figure 2-16 on page 26) or the DOS Configuration screen will appear.

The OS/2 Configuration screen enables you to modify your CONFIG.SYS file without having to directly access it. It is also used to relocate the swapper file.

Figure 2-16. OS/2 Configuration Screen



Moving the Swapper File

The swapper file can be moved during installation or afterwards from any command prompt. However, to move the swapper file during installation, the logical drive to which you are moving the swapper must already be partitioned and formatted (for example, from a previous OS/2 installation). Otherwise, you can only move the swapper file after this installation is complete and the drive is formatted.

To move the swapper file *during* installation:

1. Choose the **Select features and install** option from the OS/2 Setup and Installation screen (see Figure 2-13 on page 23). This will bring up the System Configuration screen (see Figure 2-14 on page 24).
2. After making your selections (if any), click on the **OK** button to bring up the second OS/2 Setup and Installation screen (see Figure 2-15 on page 25).
3. Select **Software configuration** from the menu bar and select **Change OS/2 parameters** to display the OS/2 Configuration screen (see Figure 2-16).

4. Indicate the new path for the swapper file in the **Swappath** entry field.
5. Click on **OK**. This will bring you back to the OS/2 Setup and Installation screen where you can then click on **Install** to continue your installation.

To move the swapper *after* installation has completed:

1. Make a backup copy of your CONFIG.SYS file.
2. Edit the CONFIG.SYS file.
   a. Locate the entry *SWAPPATH=C:\OS2\SYSTEM 2048 2048*.
   b. Change it to: *SWAPPATH=F:\ 2048 2048*, where *F:* is the last logical drive.
3. Save your changed CONFIG.SYS file.
4. Reboot the system to implement this change.
5. When you are sure the new entry is being used, delete the original SWAPPER.DAT file under *C:\OS2\SYSTEM*.

WARNING

Do not delete the *current* SWAPPER.DAT file or you will crash your system!

After making your selections, click on **OK**. This will bring you back to the OS/2 Setup and Installation screen where you can then click on **Install** to continue your installation.

When all the OS/2 diskettes have been loaded, an Advanced Options screen will display. You can add any additional device drivers from diskette and migrate your applications from other operating systems. (To migrate your applications from the Desktop, see Chapter 7, "Using the Workplace Shell Environment.")

After making your selections, click on **OK** to continue the OS/2 installation. You will be prompted to remove the final diskette and press Enter to reboot. After your system has been restarted, it will take a few minutes for the icons to appear on the Desktop.

Before you can begin accessing your other disk partitions, if you have not already done so, you should format your remaining drives at an OS/2 command prompt by typing **FORMAT x:** (where *x* is the drive letter of the drive you wish to format). Repeat this command for each drive to be formatted. Do not forget to restore any programs or data files that you backed up prior to installing OS/2 2.1.

Installing the IBM OS/2 2.x Toolkit

If your copy of the OS/2 2.x Toolkit includes update diskettes, these diskettes should be installed last because they update various components of the OS/2 2.1 operating system. Install the update diskettes prior to using the Toolkit.

To use the update diskettes, perform the following actions:

1. Close all running applications.
2. Start an OS/2 full-screen session.
3. Insert Update Diskette 1 into Drive A.
4. Change to the **A:** prompt.
5. Type **UPDATE** and press Enter. The Update program prompts for additional diskettes as required.
6. Shutdown the system and reboot.

Note that the Update program determines which diskettes are needed; not all diskettes may be required. The Toolkit is installed in the OS/2 2.1 partition by default.

NOTE

If you have a SET INCLUDE statement in your CONFIG.SYS file, you will need to modify CONFIG.SYS *after* installing the Toolkit but *before* installing the C Set/2 compiler. This is necessary because once you install the Toolkit, there will be two SET INCLUDE statements. The first statement will be the original one; the second statement will be the one created by the Toolkit installation program. Insert the information in the second statement at the beginning of the first statement, and then delete the second statement.

Use *LINK386.EXE* from the \OS2 directory instead of from the \TOOLKT20 directory.

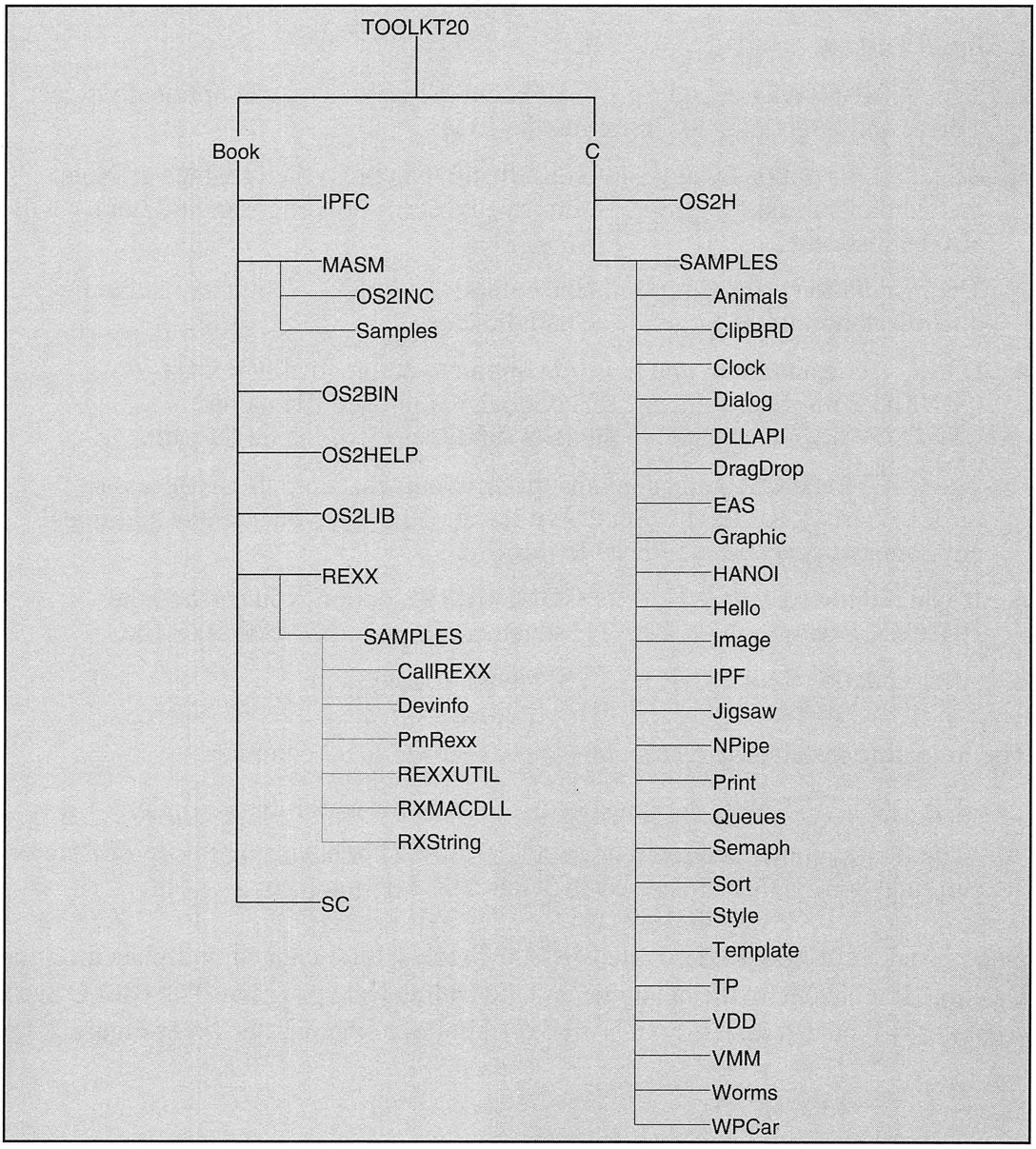
To integrate the Toolkit tools into the WorkFrame/2, perform the following actions:

1. Before installing the Toolkit, install the IBM WorkFrame/2. Ensure that the CONFIG.SYS file is updated for the WorkFrame/2 by either selecting the **Update CONFIG.SYS** checkbox or by updating it manually.
2. Reboot your computer, as specified in the WorkFrame/2 installation procedure.

3. At the OS/2 command prompt, type: addtool \toolkt20\os2bin\toolkt20.ini.

When you start WorkFrame/2 and select **Tools** from the menu bar, the entries for the five Toolkit tools will appear in a pull-down menu. Once installed, the OS/2 2.x Toolkit will have a directory tree structure similar to the one shown in Figure 2-17.

Figure 2-17. Sample of a Toolkit Directory Tree



Installing the IBM C Set/2 Compiler

The IBM WorkFrame/2 must be installed prior to installing the OS/2 2.x Toolkit and IBM C Set/2. You *must* follow this sequence or the IBM C Set/2 compiler will not work properly.

To install the C Set/2 compiler from a diskette:

1. Insert Diskette 1 in Drive A.
2. Type **A:\INSTALL** to start the installation program. You can optionally select a drive and target path for installing the IBM C Set/2.
3. Select all the options you wish to install and then press the **OK** button. Note that when you select a migration library, the corresponding standard library will also be installed.

   The installation program installs the compiler, libraries, debugger, and online documentation to the target drive and directory.
4. During the installation, you have the option to automatically update CONFIG.SYS. If you decline this option, a command file named *CSETENV.CMD* is created in the BIN subdirectory of the target path.

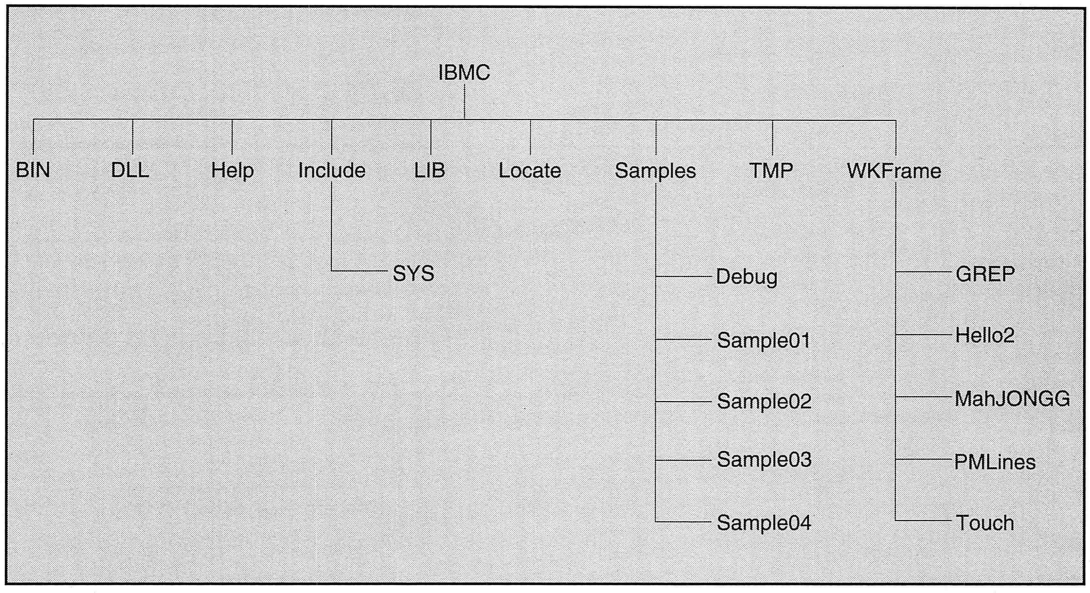
   The CSETENV.CMD file contains the environment setup that is necessary prior to starting the IBM C Set/2. We recommend that you add any additional environment variables you need to this file.
6. If you did not select the **Update CONFIG.SYS** option, you can add the *IBMC\DLL* path to the LIBPATH statement in your CONFIG.SYS file.
7. After updating your CONFIG.SYS, reboot your system.

The following installation restrictions apply to the C Set/2 compiler:

- The OS/2 LINK386.EXE must be used as the linker for this compiler.
- The floating point code will not work if you do not have a math coprocessor or an Intel 80486DX processor, which has a built-in math coprocessor.

Once installed, a new folder called *IBM C Set/2* will be created and placed on the Desktop. It will contain the debugger and the Online Help program. The IBM C Set/2 compiler will have a directory tree structure similar to the one shown in Figure 2-18.

Figure 2-18. Sample of a C Set/2 Directory Tree



If you are having problems with your installation, make sure your environment is set up properly. Use the following questions as guidelines:

- Is the directory containing *UNPACK.EXE* (which is shipped with OS/2) in your PATH statement?
- Is the HELP environment variable pointing to the *X:\OS2\HELP* directory, where *X* is your boot drive?

TIP

You should find the OS/2 2.1 installation procedure to be a straightforward process. If you are experiencing difficulty, it may be a hardware-related problem. If nothing works after your installation, try reinstalling OS/2 2.1 and ensure that the installation procedure has correctly recognized your system's components.

If all else fails, call the IBM support line (the number is provided with your OS/2 2.1 documentation).

Chapter 3.
Tips on Compiling Your OS/2 Application

OS/2 2.1 enables you to develop applications on virtually any platform that supports the Intel 80386 or higher architecture (8086/80286 or higher for DOS applications). Although the OS/2 operating system is capable of running DOS, Windows, and OS/2 16-bit and 32-bit applications, the trend is to migrate to 32-bit and take advantage of the superior 32-bit capabilities of OS/2 2.1.

OS/2 native-mode compilers exist for many of the most popular computer languages such as C, Pascal, and COBOL. A *compiler* takes source code and converts it into machine object instructions (also known as *object file* code) that can be read by the hardware platform. The object code is then linked together by a *linker* to build the executable form of the application.

To create a 32-bit executable program, you will need a 32-bit compiler and a 32-bit linker to build the application. To create an OS/2 PM-based application, you also will need a *resource compiler* to build the resources into executable form (see "Using Resource Script Files" on page 53).

The 32-bit linker and the resource compiler are provided with the OS/2 2.x Toolkit. The new release of 32-bit Borland C++ compiler for OS/2 provides its own resource compiler.

Using Compilers in OS/2 2.1

If you intend to use the advanced features of OS/2 2.1, such as the flat memory model, preemptive multitasking and multithreading, you should use a 32-bit OS/2 compiler such as the *IBM C Set/2* or *Borland C++*. (As its name suggests, the Borland compiler also supports the C++ language.) Both compilers are *ANSI C* compliant, which ensures that code that is written to ANSI standards will be portable across various platforms.

Compiling and Optimizing Your Application

The process of building an OS/2 executable file consists of compiling the source files and linking them to build the program. C Set/2 and Borland C++ are *multipass* compilers; that is, they make several passes through the files to build the machine object code from your source code.

The C Set/2 and Borland C++ compilers include a step in the compilation process called the *optimizer*. The optimizer reduces unnecessary overhead in the generated application by optimizing the code in the manner specified by the compiler options. Because of graphical user interface (GUI) environments, there is an increasing need for compilers that remove extraneous code, streamline repetitive operations, and more efficiently package program code and data. The optimizer is one means of increasing software speed and making the code more compact, thereby significantly enhancing performance.

The two most critical resources in any computer system are the CPU cycles and the availability of memory storage; therefore, a little savings in these areas can create significant gains for you as a software developer. Care must be taken, however, when using the optimizer options, as this is the area in compilers where the most compromises in code abound. For example, a particular optimization for *reducing* code instructions could subsequently *increase* data storage requirements, thereby creating a trade-off or compromised situation.

TIP

Do not use a *debugger* or generate debug information when optimization is enabled. The effects of optimization make it difficult to produce accurate debug information on optimized code. In many cases, the debug information is limited to setting break points at function entry and exit and viewing the source material at machine level. Depending on the compiler, the symbol information for the OS/2 kernel debugger could be inaccurate or simply not available.

If you decide that optimization is necessary, wait until the debug phase of the program life cycle is complete before using optimization, or generate two separate builds of your application (one for debugging and one for optimizing).

Optimizing with the IBM C Set/2 Compiler

When optimization is enabled, the C Set/2 compiler automatically inlines certain library functions such as strcpy or *memcpy* for increased performance.

Table 3-1 lists some of the IBM C Set/2 optimization options.

Table 3-1. IBM C Set/2 Optimization Options

Option	Description
/O	Enables optimization.
/O-	Disables optimization.
/G3	Optimizes code for an Intel 80386 processor.
/G4	Optimizes code for an Intel 80486 processor with downward compatibility with the Intel 80386 processor.

Optimizing with the Borland C++ Compiler

Borland optimization does not impact compilation times as much as other compiler optimizations. The optimization is more complete, and the size and speed of the optimization process can be individually controlled.

To enable the Borland optimizer, use the *-O* option on the Borland C++ compiler command line. To disable optimization, use *-Od*. Table 3-2 lists the Borland optimization options.

Table 3-2. Borland C++ Optimization Options

-O Option	Description
-O1	Optimizes for smallest code; combines -Ob, -Oc, -Oe, -Oi, -Ot, and -Oz.
-O2	Optimizes for fastest code; combines -Ob, -Oc, and -Os.
-Oa	Assumes that pointers are not aliased in evaluation or pointer subexpressions.
-Ob	Eliminates unused or dead variables.
-Oc	Optimizes common local expressions on code block with single entry and exit points.
-Od	Disables optimization except for jump-distance optimization.
-Oe	Optimizes global register allocation.
-Oi	Enables intrinsic inlining of functions.
-Os	Optimizes only code size.
-Ot	Optimizes the execution speed of your application.
-Ox	Optimizes for speed compatibility with Microsoft C optimization.
-Oz	Optimizes transformations within a function.
-O4	Optimizes Intel 80486 processors (similar to the C Set/2 /G4 option).

The inlining of functions is performed manually in the Borland C++ compiler. The intrinsic inlining process enables you to use the *-Oi* option with the *#pragma* intrinsic keyword to generate the function code. When this option is enabled, the *strxxx* and *memxxx* functions are inlined and their code is generated using the #pragma directive. For example, to use the inlining process for *memcpy*, enable the -Oi option and enter the following statement into your code:

```
#pragma intrinsic memory
```

The Borland compiler will generate the code for memcpy without calling the memcpy function.

Invoking the Linker Facility

After all source code modules have been compiled into object code, the final step towards creating an executable is to invoke the linker facility. Some compilers provide an option whereby the compiler can invoke the linker directly. The objective of the linker utility is to use the object code file(s) and run-time libraries (language bindings) to import operating system call libraries and create an executable file. The linker also handles the external references and ensures that a call from a function in one module to a function in another module is set up correctly.

Using Dynamic Link Libraries

The OS/2 operating system relies heavily on the use of *dynamic link libraries (DLLs)*, which are modules that contain executable code but cannot be run as executables. The dynamic linking process enables the size of an executable file to be greatly reduced by allowing many of the functions, resources, code, and data segments used by the executable to reside in a DLL.

An application loads a DLL only when it needs to execute a function residing in that DLL. After the DLL is loaded, it becomes available to any other program that might need it; therefore, it is only necessary to load one copy of a DLL at any given time. For example, if you are developing a large word processing application, you will most likely have several .EXE programs included in your package that perform various tasks such as text editing, spell checking, and printing. If each of these tasks requires a similar dialog box, it can be created by linking to the DLL that contains it. (This is more efficient than having to code a separate dialog box into each of your executables,

which would increase the size of your executable programs and consume more memory.) Another benefit of of using dynamic link libraries is that updating your applications becomes easier. By modifying the routines in the DLL, you can avoid having to rebuild your executable files. However, you must make sure that your modifications do not adversely affect any applications that use the DLL.

Using Header Files

For any high-level language, there are constructs upon which the language is based that you can build on to create new data structures and data types. These constructs are not usually defined in the program source code file; they are coded in *header files*, also known as *include files*.

> **TIP**
>
> Some functions, such as hook procedures, *must* be placed in a DLL. However, we recommend that you first debug your code with the functions statically linked, whenever possible (see "Using Run-Time Libraries" on page 38). This will separate any potential debug problems into manageable pieces.

You should create specific header files for your application to contain the definitions of any new data types, function prototypes (for all functions that are specific to the application), and global data variable declarations that are to be used by your application.

Header files are usually inserted into the application source code by a formal declaration in the source code. This declaration requests that the header file named in the declaration be used for resolving data references (and for function declaration information) throughout the source code file.

There are two ways that syntax for including a header file can be written:

```
#include <MYHEADER.H>
#include "MYHEADER.H"
```

The first statement includes the requested header through the mechanism of the *INCLUDE* environment variable. The second statement means that the header is to be found locally, that is, it is found in the same directory as the application being built.

If a header file included in this manner is not found in the current directory, most compilers will search through the INCLUDE environment variable definitions until the header is found.

NOTE

It is possible to override both mechanisms employed above by explicitly coding a path into the reference of the header file; however, this is *not* a recommended coding practice for portability reasons. Because development machines differ, the path might be rendered incorrect when moving your application's source code from one machine to another.

Using Run-Time Libraries

All compilers include sets of functions called *run-time libraries*, which support the common functions defined by the language supported by the compiler. These run-time libraries can provide functions for platform-independent, data-buffer management; character or string manipulation; vendor-specific extensions to the language; and standard file I/O support.

Run-time libraries are usually packaged in two forms: *statically* and *dynamically linked*. Static run-time libraries are usually referred to as *statically linked libraries*. They contain functions that are physically (statically) placed into the executable file where they perform like any other function or subroutine in the executable. Dynamically linked run-time libraries are implemented in an OS/2 DLL and have attributes similar to other system DLLs.

Applications can be linked with either the standard (*nonreentrant*) version of the language run-time libraries or the multithreaded, enabled (*reentrant*) version. These libraries also exist in both statically and dynamically linked forms to provide you with more flexible options.

Static linking implies that the C run-time function code called in an application is linked to the program in the executable module (.EXE or .DLL). Therefore, the .EXE or .DLL files will be larger because there is a copy of the run-time functions in each file where the function is called. In many cases, static linking can be faster than dynamic linking because the library functions are actually loaded with the rest of the code, thereby becoming available for use by the program more quickly. Because the

code actually resides within your program, static linking can provide simplicity as well as performance benefits. However, it can be costly in regards to resource allocation because more fixed disk space and memory is required with static linking; if you have several executable modules running concurrently, the same code is loaded into memory by each module.

Dynamic linking does not imbed the library function code into the executable module. Instead, the code for the library function is resolved at load time. Although this decreases the amount of memory needed to load the executable file and the amount of disk space required by the executable, the performance of the executable will decrease the first time the DLL is loaded for each application that uses it.

Because a resource tradeoff exists between static and dynamic linking, you must consider the size and availability of your resources before deciding whether to choose the simplicity and performance of static linking or the resource benefits gained from dynamic linking.

Accessing DLL Functions from Your Application

To access the dynamic link library functions from your application, you must create an *import library* for your DLL. The import library is used by the linker to generate a table within the executable file that can interpret the functions stored in the DLL. For example, if you are using a PM function such as WinMessageBox, that function is *unresolved* (that is, unknown to the object file) at compilation time. The import library *OS2.LIB* provides information about where and how WinMessageBox is defined. It also points to a reference to this function in a DLL (in this case, PMWIN.DLL). When the linker builds the executable form of your application, it reads OS2.LIB to find the reference to WinMessageBox. The information provided by this import library is built into a table in the executable file; when the program is loaded, the correct DLL can be located to obtain the function code for WinMessageBox.

The IMPLIB utility, which is provided with the OS/2 2.x Toolkit, is used to create import libraries. IMPLIB.EXE uses the module definition file to obtain the module names, exported functions, and other information needed to build the import library.

To create an import library, add the following expression to your make file:

```
LIBRARYNAME.LIB : MODULEDEF.DEF
   IMPLIB LIBRARYNAME.LIB MODULEDEF.DEF
```

where *LIBRARYNAME.LIB* is the name of the import library to be created and *MODULEDEF.DEF* is the module definition file to be used.

Note that import libraries are required only at link time and have no impact on program load requirements.

Creating a Development Environment

Most applications consist of several source code files, data type definitions, variable declarations, and libraries. These pieces of program code should be easily accessible to the compiler and linker so that files used in the build process can be efficiently managed. System environment variables perform this function by ensuring that all the variables needed to create the executable file are found wherever they may reside physically on the hard disk drive or other form of storage media.

An *environment variable* is simply a list of directories, where all the required pieces of code can be found. Environment variables can be placed in the OS/2 CONFIG.SYS file so that your programming environment is set every time your computer is started. These variables can be modified at any time by placing another set of environment variables in an OS/2 command (.CMD) file. This enables you to set a primary programming environment and maintain secondary programming environments, if needed. In this way, you can use an alternate compiler and set of programming tools. For example, part of a sample CONFIG.SYS file, containing environment directives for the IBM C Set/2 compiler and OS/2 2.x Toolkit, is shown in Figure 3-1.

Figure 3-1. CONFIG.SYS Environment Directives

```
REM *** Setting Programming Environment ***

SET TEMP=C:\NOWHERE
SET TMP=C:\NOWHERE
SET IPFC=D:\TOOLKT20\IPFC;
SET INCLUDE=D:\TOOLKT20\C\OS2H;D:\TOOLKT20\ASM\OS2INC;D:\IBMC\INCLUDE;
SET LIB=D:\TOOLKT20\OS2LIB;D:\IBMC\LIB;
```

These directives allow you to open an OS/2 session and compile a 32-bit program for the OS/2 Presentation Manager. If you also need to have a 16-bit programming

environment on the same hard disk, you can switch back and forth through the use of a command or batch file. For example, if the DOS batch file shown in Figure 3-2 is run in an OS/2 MVDM session, it will reset the environment to allow you to build DOS applications. This sample could also be used as an OS/2 command (.CMD) file.

Figure 3-2. DOS Batch File or OS/2 .CMD File Sample

```
RESETDOS.BAT

@ECHO OFF
CLS
PATH=D:\IBMC2\BIN;%PATH%
SET LIB=D:\IBMC2\LIB;%LIB%
SET INCLUDE=D:\IBMC2\INCLUDE;%INCLUDE%
SET TMP=D:\IBMC2\TMP
```

The percent (%) directives tell the system to maintain the current environment and append your changes to the current environment variables. In the above sample, the user's PATH environment is still intact; the D:\IBMC2\BIN setting is simply appended to it.

When modifying the environment in this manner, your changes are only valid for the *current* session, not all sessions. If the user opens another MVDM session, the path changes will be lost.

There are five main environment variables that govern the building of an application:

PATH This statement contains the directories that reference program executables, including the compiler, its associated subtools, the linker, and any other utilities required in the development environment.

For example, *ICC.EXE* is the C Set/2 compiler executable, *NMAKE.EXE* is a build tool that creates an executable by compiling and linking, and *LINK386.EXE* is the 32-bit OS/2 linker.

DPATH This statement is used to find data files such as help and message information files in OS/2 applications. (The extensions for these type of files are either *.HLP* or *.MSG*.) If the file is not found in the specified directory, the directory specified in the DPATH statement is searched. In a DOS session, the APPEND command performs the same function.

INCLUDE This statement controls the directories that contain the header files, which are usually identified by the extension *.H*. Header files contain constants, types, and prototype information. Most compilers only

search for the first instance of a header file that might be found in the INCLUDE path, so make sure that multiple copies of a particular header file do not exist.

LIB This variable specifies the directories that contain the run-time and import library files. Most linkers only search until the first instance of a library file is found along the path specified, therefore, it is important that you do not have any duplicate .LIB files and that your LIB path contains only the required directories.

TMP This variable is commonly referred to as the compiler's *working directory*; that is, it specifies the directory where the compiler places its temporary working files during the compilation process. Most compilers erase the contents of this directory when they are through with the temporary files. Therefore, this directory should appear empty after compilation. If this environment variable is not set, the compiler uses the current directory as the temporary working directory.

TIP

If your application encounters errors during the building process or does not behave correctly during execution, check for incorrect header or library files.

IBM C Set/2 provides a sample command batch file called *CSETENV.CMD*, which contains the environmental directives that set up the environment so that the compiler can be used. Borland's *Integrated Development Environment (IDE)* provides a notebook-like control panel where these variables can be set up and stored for later use.

Supported Calling Conventions

The C language is designed to be portable across many platforms. However, to handle the limitations or shortcomings of some architectural platforms, certain implementation-specific constructs must be defined and included in compilers. The OS/2 2.x operating system is currently an Intel-based operating system that is backwards-compatible with OS/2 1.x 16-bit applications, which were designed around the Intel 80286 architecture. Because of this, certain limitations from these older designs (for example, the segmented memory model) created the need for provisions (such as calling conventions and function definition modifiers) to be included with the C Set/2 and Borland C++ compilers for ANSI standard compliance.

TIP

If your application contains more than one source module, all functions that are declared in one module but are also used by other modules should be prototyped in header files. The appropriate header file should be included in any source module that uses the functions. This will ensure that the declaration and calling sequence of common functions remains consistent.

IBM C Set/2 Calling Conventions

The IBM C Set/2 compiler provides five calling conventions: two complete 32-bit calling conventions (*optlink* and *system*) and three 16-bit calling conventions (*far16 cdecl, far16 pascal*, and *far16 fastcall*), which are usually referred to without their far16 prefix. These conventions enable 32-bit code to functionally coexist with 16-bit code. The cdecl and pascal conventions are equivalent to the cdecl and pascal used in the 16-bit C/2 compiler or Microsoft C compiler. The fastcall convention provides compatibility with the fastcall of the Microsoft C 6.0 compiler.

With the C Set/2 compiler, you can specify the calling convention in one of two ways:

- By specifying a calling convention universally for all functions within a program using the compiler options, /Mp or /Ms. The */Mp* option uses the default optlink calling convention. (To use the OS/2 APIs, you must include the Toolkit headers.) The */Ms* option uses the alternate-system calling convention. (To call functions, you must include the necessary library header files.)
- By using the linkage keyword *#pragma* to specify linkage for individual functions within your program. The #pragma linkage takes precedence over the compiler options if both methods are used.

The following calling conventions are supported by the IBM C Set/2 compiler:

optlink The fastest 32-bit calling convention and the default calling convention of the compiler. The parameters are pushed onto the stack from right to left. The caller is responsible for cleaning up parameters on the stack.

Note that the values in the 32-bit base addressing register (EBX), destination index register (EDI), and source index register (ESI) are maintained across the function call. The accumulator register (EAX), the count register (ECX), and Data I/O register (EDX) are not maintained.

Conforming parameters, which are passed in the EAX, ECX, and EDX registers, consist of 1-, 2-, and 4-byte signed and unsigned integers, enumerations, and all pointer types. If you are using conforming parameters extensively, they should appear as far to the left as possible in your formal parameter list and follow each other directly so that they will be considered first for registers. This will speed up the processing of the parameter list.

system A 32-bit calling convention that is similar to the standard C convention. All parameters from a function call are passed on the 80386 stack space and are pushed onto the stack from right to left. The calling function is responsible for cleaning up the parameters on the stack, which are 4-byte aligned.

cdecl The 16-bit, standard C convention. All parameters from a function call are pushed onto the stack from right to left. The caller, upon return from the function call, cleans up the stack of all data passed and returned. (Note that this calling convention is nonreentrant and should not be used for multithreaded programming.)

pascal This 16-bit calling convention supports calling functions implemented in other high-level languages such as Pascal. The parameters placed on the stack are passed from left to right.

The pascal convention is used primarily for the OS/2 Presentation Manager because it provides two basic advantages to the user: compact code and speed. The code generated is less (smaller) in size because the fixed number of parameters is known at compile time. It is also faster because it is the responsibility of the *callee* to clean up the stack prior to returning to the caller.

fastcall This 16-bit calling convention is similar to optlink in that it uses three registers to take parameters. Parameters of type *char* and *unsigned char* are stored in the AL, DL, and BL registers; *short* and *unsigned short* are stored in AX, DX, and BX; and *long* and *unsigned long* are stored in DX and AX. All other parameters are pushed onto the stack from left to right.

Note that a function compiled with a particular calling convention cannot be called by a different calling convention. For example, if a function is compiled with the *optlink* convention, you cannot call it with the *system* convention.

TIP

For greater portability and usability, we recommend that you include function prototypes for *all* functions.

Borland C++ Calling Conventions

The following calling conventions are supported by the Borland C++ compiler:

cdecl The 16-bit, standard C convention. All parameters from a function call are pushed onto the stack from right to left. The caller, upon return from the function call, cleans up the stack.

pascal This 16-bit convention supports calling functions implemented in other high-level languages such as Pascal. The parameters are pushed onto the stack from left to right. In this convention, it is the responsibility of the *callee* to clean up the stack prior to returning to the caller.

fastcall This 16-bit calling convention is similar to optlink in that it uses three registers to take parameters. Parameters of type *char* and *unsigned char* are stored in the AL, DL, and BL registers; *short* and *unsigned short* are stored in AX, DX, and BX; and *long* and *unsigned long* are stored in DX and AX. All other parameters are pushed onto the stack from left to right.

stdcall This 16-bit calling convention is used for forcing the standard OS/2 argument-passing convention. Parameters are pushed onto the stack from right to left. The *callee* is responsible for cleaning up the stack.

This convention is solely provided for compatibility; that is, it allows support for older 16-bit code that might be too impractical to rework for 32-bit.

Two keywords, APIENTRY and EXPENTRY, are used to describe function definitions. *APIENTRY* defines the type of function that indicates an API entry point in a DLL. *EXPENTRY* defines dialog procedures and indicates that the function is to be exported.

These keywords are used in the following manner:

```
MRESULT EXPENTRY StandardDlgProc(hwnd, msg, mparam1, mparam2);
MRESULT APIENTRY StandardDllProc();
```

The Borland C++ compiler uses the *precompiled header*, which is designed to decrease compilation time. A compiler can use a lot of time parsing header files and searching for used declarations and definitions. When this information is found, it is placed into a compiler *symbol* file. If you have several source files that normally use the same include file, the symbol file will be parsed several times (as many times as the file is included) and needlessly create the same symbol table over and over again. Borland C++ reduces this process by creating the symbol table on the first compilation. The table is stored in the file *TCDEF.SYM* in the same directory as the compiler. The next time a declaration is needed, the symbol file is loaded directly from the hard disk, which reduces compilation for large source files.

OS/2 Programming Models

The design of the OS/2 operating system allows downward executable compatibility for either pure 16-bit applications or a mixture of 16-bit and 32-bit application support. The following lists show the supported programming models available to you in OS/2 2.1.

Pure 16-Bit Applications

- Can run in OS/2 1.x or OS/2 2.x environments
- Use 16-bit C run-time libraries
- Can call 16-bit dynamic link libraries
- Use 16-bit OS/2 APIs
- Are linked with a 16-bit linker, such as LINK.EXE
- Compile with a 16-bit C compiler, such as Microsoft C 6.0 or IBM C/2 v1.1.

Mixed 16-Bit Applications

- Can only run in OS/2 2.x (32-bit) environments
- Use 16-bit C run-time libraries
- Can call 16- or 32-bit dynamic link libraries
- Use 16-bit OS/2 APIs
- Are linked with a 32-bit linker, such as LINK386.EXE
- Compile with a 16-bit C compiler, such as Microsoft C 6.0 or IBM C/2 v1.1.

Pure 32-Bit Applications

- Can only run in OS/2 2.x (32-bit) environments
- Use 32-bit C run-time libraries

- Can call 16- or 32-bit dynamic link libraries
- Use 32-bit OS/2 APIs
- Are linked with a 32-bit linker, such as LINK386.EXE
- Compile with a 32-bit C compiler, such as IBM C Set/2 or Borland C++.

Mixed 32-Bit Applications

- Can only run in OS/2 2.x (32-bit) environments
- Use 32-bit C run-time libraries
- Can call 16- or 32-bit dynamic link libraries
- Use either 16-bit or 32-bit APIs in a source file, but not both
- Are linked with a 32-bit linker, such as LINK386.EXE
- Compile with a 32-bit C compiler, such as IBM C Set/2 or Borland C++.

About the Thunking Process

Thunking is the mechanism that enables compatibility between 16-bit and 32-bit code. The OS/2 2.x operating system contains a thunking subsystem that is transparent to application developers. This subsystem enables you to call 32-bit APIs that convert, or *thunk,* parameters to their 16-bit API equivalents. (A thunk process enables a 16-bit application to call 32-bit code or a 32-bit application to call 16-bit code.)

If you use 16-bit API sets such as *KBDxxx* or *VIOxxx* in OS/2 1.x, the thunking process will allow you to continue using these functions in OS/2 2.x. The 32-bit OS/2 compilers provide for thunking and use header files and translations to simplify the process.

The basic data types used by 16-bit code are WORDs and SHORTs; basic data types used by 32-bit code are DWORDs (double words) and LONGs. The OS/2 thunking subsystem ensures compliance by converting the length difference between these data types, as needed, and converting pointers and the stack.

IBM C Set/2 and Borland C++ compilers allow 32-bit applications to call a 16-bit API function and use a 16-bit far pointer (16:16) from the flat memory model employed by the 32-bit application. This compatibility is achieved through the use of *keywords* (see "Using Module Definition Files" on page 56).

About Memory Allocation

The OS/2 memory management features enhanced memory allocation. For example, system memory is provided as a 4GB linear address space. However, an application can access only 512MB of of this linear space. Because there is no longer a need for direct segment manipulation or pointer calculations, application performance and 32-bit porting processes are greatly improved.

Access to system memory is through the allocation of *memory objects*. Each memory object consists of a set of one or more 4KB *pages*, which can be individually swapped to and from the hard disk. This is useful when there is not enough system memory available to load the entire application or memory object.

Alternative Compiler Options

Two types of alternative options are provided by compilers: *calling convention* options and *listing/source code* options.

Calling Convention Options

A compiler can supply several options for declaring the default calling convention to be used when building a source code module.

If you use the IBM C Set/2 compiler options, the calling convention specified by the option flag will be used *globally* throughout the source module you are building. Any function that you wish to employ with a different calling convention must *explicitly* define the function prototype and function declaration with the calling convention that is to be used.

Some of the more important command line options for the IBM C Set/2 compiler are as follows:

/Mp Specifies *optlink* calling conventions.

/Ms Specifies *system* calling conventions.

Some of the more important command line options for Borland C++ compiler are as follows:

-pc Uses the *cdecl* calling convention throughout.
-p Uses the *pascal* calling convention throughout.
-pr Uses the *fastcall* calling convention throughout.
-p- Uses the *stdcall* calling convention throughout. This is the default.

Listing/Source Code Compiler Options

Some of the more important listing options for the IBM C Set/2 compiler are as follows:

/Fa Specifies that an assembler listing file can be generated with the source code included in the output file as comments. This option can be used to determine errors when data objects are not being manipulated in the correct manner. For example, the assembler output could reveal that a field in a data structure is being erroneously used as a WORD-sized (16-bit) field instead of a DWORD-sized (32-bit) field as defined in the structure definition.

The default for this option is */Fa+*, which generates an output file with an *.ASM* extension to the file name. Using the */F-* option inhibits the generation of this output file. Using */FaNAME* enables you to substitute a different name to be output (instead of the name of the source module) in the NAME field of the */Fa* option.

/Fm Specifies that a linker map file is to be generated. The default */Fm-* is to *not* generate such a file. Using */Fm+* will generate a map file with the default extension of .MAP. A map file is useful for determining problems in physical code layout. For example, misaligned code or data can cause serious performance implications for your application. Using a map file assists in the process of determining the proper code layout.

Note that the */Fm* options assume that the linker utility is invoked directly from the compiler and that the output from the compiler passes directly to the linker. Therefore, these options should *not* be used with the */C* (compile only) flag specified on the command line.

Source code options for the IBM C Set/2 compiler are shown in Table 3-3 on page 50 with a brief description of what the option does and its default status.

Table 3-3. IBM C Set/2 Compiler Options

Switch	Description	Default
/Sa	Allows ANSI standard conformity.	
/Sd-	Sets the default file extension as .OBJ for linking purposes.	YES
/Sd+	Sets the default file extension as .C for compilation purposes.	
/Se	Uses C Set/2 language extensions.	YES
/Sh-	Does not use data definition names (*ddnames*).	YES
/Sh+	Uses data definition names.	
/Sm	Uses migration extensions.	
/Sn-	Does not use double-byte character set (DBCS).	YES
/Sn+	Uses double-byte character set (DBCS).	
/Sp(*x*)	Aligns structures on *x*-byte boundaries, where *x* = 1, 2, or 4.	/Sp4
/Sq	Uses no sequence numbers.	YES
/Sq[l,r]	Specifies left- and right-sequence values.	
/Sr-	Uses new type conversion rules (ANSI conformance).	YES
/Sr+	Uses old type conversion rules (non-ANSI conformance).	
/Ss-	Does not allow use of double slashes (//) for comments.	YES
/Ss+	Allows use of double-slash comments.	
/S2	SAA Level 2 conformity.	

The source code options for the Borland compiler are shown in Table 3-4 with a brief description of what the option does and its default status.

Table 3-4. Borland C++ Compiler Options

Switch	Description	Default
-A	Forces strict ANSI compliance.	
-AK	Uses Kernighan and Ritchie keywords.	
-AT	Uses Borland C++ keywords.	
-AU	Uses UNIX keywords.	
-C	Uses nested comments.	

Using the NMAKE Utility

The *NMAKE* utility, which is provided with the OS/2 2.x Toolkit, helps automate the application build process by allowing you to merge the compile and link processes. NMAKE can be used to examine which files have changed since the last compilation, and to build only the new changes into the executable file.

NMAKE uses a simple ASCII text file (marked by several keywords and macros) as its input and executes the instructions needed to build the program. This text file is called a *make file*, which has a .MAK extension. Depending on your development environment, you can either have a single file called *MAKEFILE*, or you can name your make file the same as your target program.

If you choose the first method, you need only to invoke *NMAKE* from the command line because it is assumed that your file name is MAKEFILE. If you have named your make file (for example, SAMPLE.MAK), NMAKE must be invoked with the file name as a parameter (for example, *NMAKE SAMPLE.MAK*).

The make file consists of a series of build directives that invoke the compiler and linker, along with inference rules and command macros that instruct NMAKE to build the executable in accordance with your specified compile and link options. Comments can be inserted into the make file through the use of the # symbol. These comments are only for the programmer's benefit and are ignored by the NMAKE utility.

The format for a build directive is as follows:

```
target file : dependency files
   build command
```

The *target file* is the file that you want the build process to produce. The *dependency files* are required to build the target file. The *build command directive* instructs the proper build tool to invoke the build process. For example, the command-line options for the compiler and linker are specified in the build directive.

NMAKE also uses a series of *inference rules*, which decide what file types are needed to build a target file from a specified file type. The format for inference rules is to start with a period (.), follow with the dependency file extension, the target file extension and a colon (:), and end with the build command directive (as shown below):

```
.dependency file extension.target file extension:
    build command
```

Using the rules of inference, the compiler can be instructed to build object (.OBJ) files from source .C files as shown in the following example:

```
.c.obj:
   ICC  $(cflags) $*.c
```

In this case, *cflags* represents a command macro that indicates to the C Set/2 compiler which command line options are needed to build the source file into object code. Two simple predefined macros exist that can help you in building your application:

$* Indicates that the target file name is to be substituted without an extension.
$** Indicates that the entire list of dependency files are to be substituted.

If you need to build multiple target files, you can use the *ALL* keyword. This keyword is used as your build directive; its dependent target files become its dependency files as shown in the following example:

```
ALL : SAMPLE1.EXE SAMPLE2.EXE
```

The sample make file (MAKEFILE) in Figure 3-3 uses the build directive, inference rules, and the ALL keyword.

Figure 3-3. Sample Make File for a PM Control Demo

```
CFLAGS  = /c /Se /Ss /Kc+ /Kp+ /Kb+ /Ti+ /DDEBUG
LFLAGS  = /CO /NOD /NOE /BASE:0x10000 /ALIGN:4
CC      = ICC
LIBS    = os2386.lib dde4mbs.lib
HELPIPF = magicw.ipf magicw1.ipf magickw.rc

# TARGET...
all : magickw.exe

# Resources
magickw.res : magickw.rc   menures.h   magicdlg.h   testdlg.h    helpids.h \
              helpmenu.rc1 testdlg.dlg magicdlg.dlg magichlp.rc1 msghndlr.rc1 \
              help.dlg
  rc -r magickw.rc

# Dependencies...
magickw.obj   : magickw.c
  $(CC) $(CFLAGS) magickw.c
magicwp1.obj : magicwp1.c menures.h magicdlg.h testdlg.h notebook.h listbox.h \
               dlgbox.h
  $(CC) $(CFLAGS) magicwp1.c
magicdp1.obj : magicdp1.c magickw.h magicdlg.h
  $(CC) $(CFLAGS) magicdp1.c
magicdp2.obj : magicdp2.c testdlg.h
  $(CC) $(CFLAGS) magicdp2.c
msghndlr.obj : msghndlr.c msghndlr.h
  $(CC) $(CFLAGS) msghndlr.c
syshelp.obj   : syshelp.c syshelp.h helpids.h
  $(CC) $(CFLAGS) syshelp.c
magickw.exe   : magickw.obj  magicwp1.obj magicdp1.obj magicdp2.obj syshelp.obj \
                msghndlr.obj magickw.res
  link386 @magickw.lnk;
  rc magickw.res magickw.exe
# Build IPF help file for Magic Window (compile to an intermediate file first)
magicw.hlp : $(HELPIPF)
  icc -P magicw.ipf
  ipfc magicw.i
```

Resource Script Files

The primary advantage of using a graphical user interface (GUI) programming environment is that you have access to predefined graphical resource objects such as dialog boxes, menus, and colorful icons. These objects are collectively known as *resources* and can be manipulated by different applications. Because of this, they are maintained independently from the program source code in a file called the *resource script file (.RC).*

Several tools assist in the creation and manipulation of program resources. For example, dialog boxes can be created and tested with the Dialog Editor; icons can be created with the aid of the Icon Editor. (Both of these editing utilities are provided with OS/2 2.x Toolkit.) Resources such as bit maps can be obtained from third party applications or scanned in directly from a picture with the use of a scanner and scanning program. Other resources such as string tables and accelerator keys are hand coded into the resource script file.

A resource script file is simply a text file that is compiled by a resource compiler into an object file, which is eventually built into an executable file or a DLL. The resource script file has an extension of *.RC*; after compilation, it has the extension *.RES*.

To invoke the resource compiler, add the following statement to your make file:

```
RC -R SAMPLE
```

This command assumes the file name SAMPLE has an extension of .RC; it builds the resource object code into a binary file with the extension .RES. The resources are then merged into the executable file by invoking *RC SAMPLE.RES* in your make file.

To combine both of these steps, invoke the RC command with the input resource script file and output executable file, as follows:

```
RC SAMPLE.RC SAMPLE.EXE
```

To use this method to build resources into a dynamic link library (DLL), invoke the RC command as follows:

```
RC SAMPLE.RC SAMPLE.DLL
```

Figure 3-4 shows how to use the Dialog Editor to create the dialog template and include the template in the resource file through the use of the *rcinclude* keyword.

Figure 3-4. Resource File Sample 1

```
#include <os2.h>
#include "menures.h"          /* Menu and message identifiers     */
#include "syshelp.h"          /* Message-handler/help identifiers */
#include "magicdlg.h"         /* Dialog identifiers               */
#include "testdlg.h"          /* Dialog identifiers               */

POINTER ID_MAGICK MAGICK.ICO  /* Main Menu                        */
MENU ID_MAGICK
   BEGIN
     SUBMENU "~Options", IDM_OPTIONS, MIS_TEXT
     BEGIN
       MENUITEM "~Notebook",        IDM_OPTIONS_NOTEBOOK, MIS_TEXT
       MENUITEM "~Details",         IDM_OPTIONS_DETAILS,  MIS_TEXT
       MENUITEM "~Listbox",         IDM_OPTIONS_LISTBOX,  MIS_TEXT
       MENUITEM "~Message box",     IDM_OPTIONS_MSGBOX,   MIS_TEXT
       MENUITEM "~Dialog box",      IDM_OPTIONS_DLGBOX,   MIS_TEXT
       MENUITEM "~Other controls", IDM_OPTIONS_CONTROLS, MIS_TEXT
       MENUITEM SEPARATOR
       MENUITEM "E~xit \tF3",       IDM_OPTIONS_EXIT_F3,  MIS_TEXT
     END
     SUBMENU "~Help", ID_HELP_SUBMENU_1
       rcinclude helpmenu.rc1
   END

ACCELTABLE ID_MAGICK          /* Accelerators (F3 = Exit)      */
   BEGIN
     VK_F3, IDM_OPTIONS_EXIT_F3, VIRTUALKEY
   END

STRINGTABLE                   /* Message string table          */
   BEGIN
     IDS_ERR_WINDOW_CREATE, "Window creation failed"
     IDS_ERR_WINDOW_POS,    "Window positioning failed"
     IDS_MENU_TITLE,        "Magic Window Sample Application"
     IDS_USR_MSG1,          "Example of \x0D displaying two lines"
     IDS_USR_MSG2,          "Cannot create control"
     IDS_USR_MSG3,          "Storage allocation error \x0D rc=%"
   END

rcinclude testdlg.dlg         /* Dialog Box template            */
rcinclude magicdlg.dlg        /* Dialog Box template            */
rcinclude msghndlr.rc1        /* Message-handler string table   */
rcinclude help.dlg            /* "About" help dialog template   */
```

An alternative method of creating the dialog boxes is to use a sample resource script file where all resources are hand coded directly (see Figure 3-5).

If the application that you are developing relies on using many resources and your resource script file has grown very large, we highly recommend you keep your dialog boxes separate by using the *dlgtemplate (.DLG)* file.

Figure 3-5. Resource File Sample 2

```
#include <os2.h>
#include "sample.h"

SOURCE   IDT_TEXT IDT_HLP   helpfile.txt    /* Resource for ASCII text file   */
POINTER  ID_WINDOW          sample.ico      /* Resource for program ICON      */
BITMAP   IDB_PROD           prodinfo.bmp    /* Resource for bit map in dialog */

MENU     ID_WINDOW
BEGIN
   SUBMENU "~File", ID_FILE_MENU
   BEGIN
     MENUITEM "Open a File\tF5",          IDM_OPENFILE,  MIS_TEXT
     MENUITEM "Open a Metafile\tF6",      IDM_CLOSEFILE, MIS_TEXT
     MENUITEM SEPARATOR
     MENUITEM "Product Information\tF2",  IDM_ABOUT,     MIS_TEXT
     MENUITEM "Exit Program\tF3",         IDM_EXIT,      MIS_TEXT
   END
   SUBMENU "~Options", ID_OPTIONS_MENU
   BEGIN
     MENUITEM "Play a Showtune\tESC",     IDM_PLAYTUNE,  MIS_TEXT
     SUBMENU "Change Colors",             IDM_COLORS,    MIS_TEXT
       BEGIN
         MENUITEM "RED",                  IDM_RED,       MIS_TEXT
         MENUITEM "BLUE",                 IDM_BLUE,      MIS_TEXT
         MENUITEM "GREEN",                IDM_GREEN,     MIS_TEXT
       END
   END
   MENUITEM "F1=Help", ID_HELP, MIS_HELP | MIS_BUTTONSEPARATOR
END

STRINGTABLE                     /* Message string table used by WinMessageBox */
{                               /* to display message in .C file.             */
   IDS_CLASS,     "Sorry, that is incorrect"
   IDS_TERMINATE, "Are you ready to exit the program?"
}

ACCELTABLE ID_WINDOW PRELOAD        /* Menu item accelerator keys; equal to a */
{                                   /* keystroke for each menu item IDM_.     */
   VK_ESC,  IDM_PLAYTUNE,   VIRTUALKEY
   VK_F2,   IDM_ABOUT,      VIRTUALKEY
   VK_F3,   IDM_EXIT,       VIRTUALKEY
   VK_F5,   IDM_OPENFILE,   VIRTUALKEY
   VK_F6,   IDM_CLOSEFILE,  VIRTUALKEY
}

MESSAGETABLE LOADONCALL   /*Similar to Stringtable except used for FATAL errors*/
{
   MSG_COULDNT_OPEN_FILE,   "Unable to open file format"
   MSG_FILE_NOT_FOUND,      "File Not Found!"
   MSG_ERROR_ALLOC_MEM,     "Error allocating memory for bit map file!"
}

DLGTEMPLATE ID_ABOUTDIALOG LOADONCALL MOVEABLE DISCARDABLE
{
   DIALOG "", 0, 80, 130, 250, 120,,FCF_DLGBORDER
   {
      CTEXT "SAMPLE - Sample Resource File Program"  -1, 45, 90, 180, 8
      CTEXT "by"                                     -1, 45, 80, 180, 8
      CTEXT "The Authors"                            -1, 45, 70, 180, 8
      CTEXT "(C) Technical Productions, Inc., 1993"  -1, 45, 55, 180, 8
```

Resource File Sample 2 (continued)

```
      CTEXT "Beta Drop 1.23(s) 12/31/92" ,          -1, 45, 45, 180, 8
      DEFPUSHBUTTON "OK"    DID_OK,                100,  8, 60,  16, WS_GROUP
   }
}

DLGTEMPLATE ID_PRODINFO LOADONCALL MOVEABLE DISCARDABLE
{
   DIALOG "SAMPLE - Product Information",
   ID_PRODINFO,
   75, 75, 275, 130,
   FS_NOBYTEALIGN | WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
   {
      CONTROL IDB_PROD, IDB_PROD, 110, 70, 21, 21, WC_STATIC, SS_BITMAP | WS_GROUP
      CTEXT          "(C) TPI, 1993. All Rights Reserved.", 256, 7, 54, 255, 8
      DEFPUSHBUTTON  "~OK"  DID_OK,    111, 8, 45, 15, WS_GROUP
   }
}
```

As shown in Figures 3-4 and 3-5, every resource must have a unique identifier. Each resource attribute, such as a dialog window, push button, or entry field, must also have a unique identifier. The identifiers are defined in the include file *SAMPLE.H* that is included in the resource script file.

Module Definition Files

A *module definition (.DEF)* file is a text file that is linked with the executable (.EXE) file. The purpose of the .DEF file is to set flags and other information into the executable file so that the OS/2 operating system can differentiate between DOS and OS/2 applications. The module definition file is also used to set the required HEAPSIZE and STACKSIZE and to tell the operating system whether to load the code immediately or only when called.

An example of a module definition file is shown in Figure 3-6.

Figure 3-6. Example of an OS/2 Module Definition File

```
NAME          MAGICKW WINDOWAPI              ;Program name and type; WINDOWAPI
                                             ;indicates that this is a PM program.
DESCRIPTION   'Magic Window'
STUB          'OS2STUB.EXE'                  ;Use OS/2's default stub program.
CODE          PRELOAD MOVEABLE               ;Make code segment movable.
DATA          LOADONCALL MOVEABLE MULTIPLE
HEAPSIZE      32768                          ;Allocate a 32KB heap size.
STACKSIZE     8192                           ;Allocate an 8KB stack size.
EXPORTS       MagickWndProc       @1         ;Export-required window procedures
              MagicDlgMsgProc     @2         ;and functions.
              CtrlDlgMsgProc      @3
```

Module definition file keywords are shown in Table 3-5.

Table 3-5. Module Definition File Keywords

Keyword	Description
NAME	Describes the module name (which is usually the same as the application's executable name) and the executable file's operation mode (which indicates whether your application is an OS/2 PM-based, a VIO windowable, or a VIO full-screen application). *WINDOWAPI* indicates that this is a Presentation Manager application. (If this parameter had been *WINDOWCOMPAT*, it would have indicated that this is not a PM application but it can run in a text window in a PM session.)
DESCRIPTION	Contains comments inserted into the executable file; usually used to embed program or copyright information into the application.
STUB	Optional (can be helpful if a user attempts to execute your application from a command line that cannot run the application). This keyword inserts a stub into the executable file that causes an error message if an attempt is made to run the application in the wrong mode, such as running an OS/2 application in a DOS environment. (OS/2 2.1 provides a default stub executable that warns the user when an attempt is made to execute an OS/2 application in native DOS.)
CODE	Describes the function of your application's code segment. The available options are PRELOAD, MOVEABLE, and DISCARDABLE. The *PRELOAD* option indicates that the segment is loaded into memory when the application is invoked. *MOVEABLE* indicates that the code segment can be moved to another location to concatenate available memory. *DISCARDABLE* indicates the segment can be discarded and reloaded from the executable, if necessary.
DATA	Describes the function of your application's automatic data segment. Some of the available options are PRELOAD, MOVEABLE, MULTIPLE, and SINGLE. *PRELOAD* indicates that the segment is loaded into memory when the application is invoked. *MOVEABLE* indicates that the code segment can be moved to another location to concatenate available memory. *MULTIPLE* indicates that an individualized copy of the data segment is created for each process that use the DLL. (This is the default.) *SINGLE* indicates that only one data segment is created for all processes that use the DLL.
HEAPSIZE	Indicates how much memory is available for use in the data segment of your application.
LOADONCALL	Indicates that the data segment is loaded into memory whenever it is required. This is the default.
PRELOAD	Indicates that the data segment is loaded into memory whenever it is required.
READONLY	Indicates that only one data segment is created for all processes that use the DLL.
READWRITE	Indicates that an individualized copy of the data segment is reserved for all processes that use the DLL. This is the default.

Module Definition File Keywords (continued)

Keyword	Description
STACKSIZE	Indicates how much memory is available in the run-time stack for use by your application. If you plan to use huge arrays locally (or if you frequently use recursive functions or large nonstatic variables), you will need a large stack size to accommodate these variables.
EXPORTS	Exports these functions so that Presentation Manager can use them; that is, so that they can be called from another module (in this case, Presentation Manager).
IMPORTS	Lists functions whose linkage must be resolved by OS/2 2.1 when dynamically loading modules. These functions will often be in DLLs.
PROTMODE	Optional. Indicates that the application can only be run in *protected mode*. If this parameter is included, the .EXE file will be smaller.

Chapter 4.
Tips on Printing in OS/2 2.1

The OS/2 2.1 print subsystem has better printing control than previous versions. OS/2 applications use the system printer drivers provided with OS/2 2.1. They can also use multiple printer drivers through one or more print queues. Each application can have different job properties.

Single-threaded programs, such as WIN-OS2 or DOS applications, can print to the OS/2 spooler to take advantage of OS/2's multitasking feature by queuing up several jobs simultaneously. Windows applications use WIN-OS/2 printer drivers; DOS applications use their own printer drivers through the OS/2 spooler system.

The OS/2 print subsystem consists of the spooler object and the printer object. All other print-related objects can be accessed through the printer object.

Spooler Object

The *spooler object* (see Figure 4-1) manages all the jobs pending on the system. It can be accessed through the OS/2 System folder.

Figure 4-1. Example of the Spooler Object



The OS/2 spooler maintains all the spooled job files that are waiting for an available printer or port. These jobs can come from OS/2, DOS, or WIN-OS/2 applications.

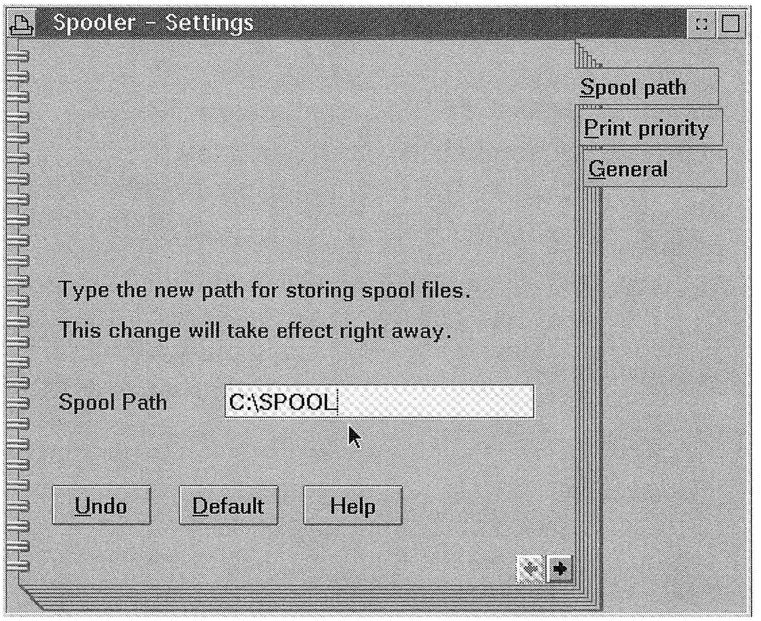
Spooler Space Requirements

Spooler space requirements increase if many print jobs or large jobs are spooled. Depending on the size of your spool file, a message box might appear with a warning about insufficient spooler space. Undetermined errors can be caused by insufficient

spooler space in the SPOOL directory. (Approximately 10 to 15MB of spooler space is usually sufficient for several pages of graphics output.)

The SPOOL directory is created during OS/2 installation. This directory is located in the same partition as the OS/2 operating system. A printer queue is placed in a subdirectory under the \SPOOL path; its name consists of the first 8 characters (with no blanks) of the name of the associated printer object. If you print large jobs frequently, we recommend you set up separate storage or disk space for the spooler. To change the path of the SPOOL directory to this storage area, open the Spooler folder located in the OS/2 System folder. This will display the Settings notebook for the spooler object (see Figure 4-2). Type the new path of your SPOOL directory.

Figure 4-2. Spooler Settings Notebook



You can enable or disable the spooler by clicking on the title bar of the Spooler Settings notebook and selecting the **Enable spooler** option or the **Disable spooler** option (if the spooler is already enabled). It is not necessary to restart your system to begin using the spooler; however, you must reboot your system if you have disabled the spooler. You cannot disable spooling to a network printer on a network server.

When the spooler is disabled, the print jobs cannot be viewed and the *direct printing* mode is used. The print jobs go directly to the output devices, bypassing the spooler.

TIP

Direct printing should be used when only one print job is sent at a time. If this mode is used for multiple print jobs, the data from the different print jobs could be mixed.

Generally, *queued printing* is the method used for multitask printing in OS/2 2.1. All print jobs are managed by the spooler, which ensures that the output device is properly shared by all applications. Enabling the spooler allows the queued files to be printed.

Standard and Raw Data Spooling

Spooled file data in the OS/2 spooler queues is always in one of two formats: *PM_Q_STD* (standard data) or *PM_Q_RAW* (raw data). PM_Q_STD is also known as a *metafile* format. PM_Q_RAW is a printer-specific format.

PM_Q_STD or *standard* spooled files record graphic orders and attributes in a packed binary format; this makes the spooled files much smaller than the raw data files. Standard spooled files are *device independent*, which means that the files can be sent to different devices. If you work in a LAN environment, using the PM_Q_STD format to send the spooled file to the network server will significantly reduce network traffic.

PM_Q_RAW spooled files contain output to be sent to the printer. This format, which is the same as for the control command file, is *printer dependent*. For example, spooled files generated in PM_Q_RAW format can consist of PPDS language for IBM 4019 printers, PCL language for LaserJet printers, or PostScript language for PostScript printers.

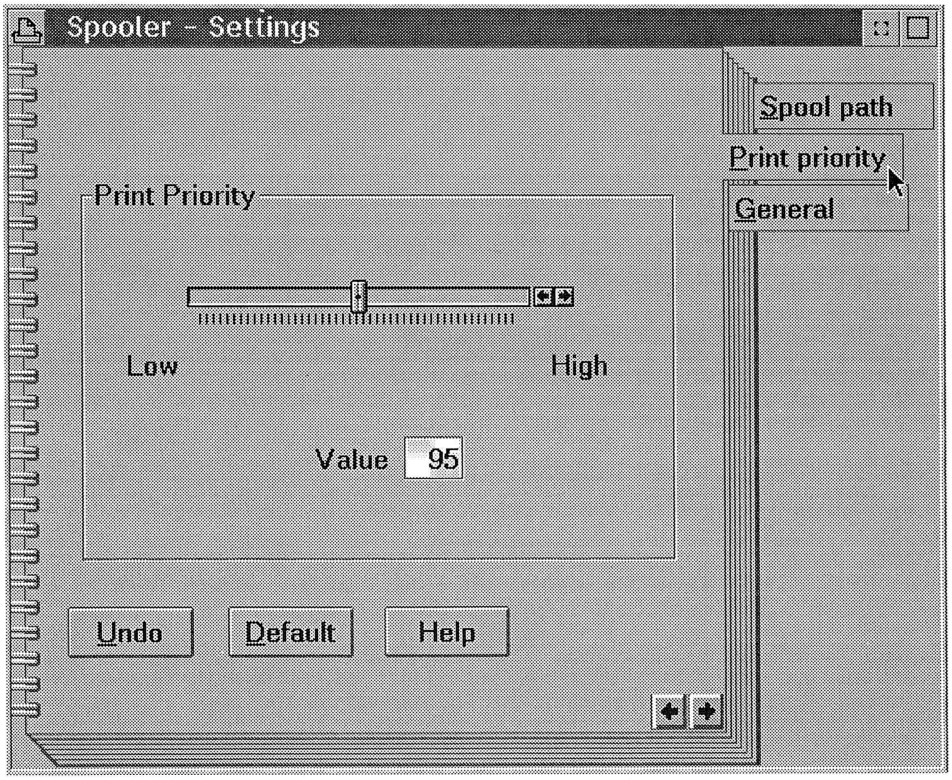
WIN-OS/2 and DOS applications use the PM_Q_RAW format as do some OS/2 applications if they print complicated graphics. However, PM_Q_STD is used for most OS/2 application spooled files.

Setting Print Priority

Print priority is a new setting added to the spooler in OS/2 2.1 (see Figure 4-3 on page 62). With this feature, you can adjust the rate at which spooled print jobs are printed. The valid values range from *1* (lowest) to *189* (highest) with a default setting of *95*. Setting the print priority higher will give your print jobs a higher priority but may slow the response time of the system, depending on how you are using your system (for

example, your mouse may respond more slowly). We recommend setting the value higher when using a printer server and lower when running multiple applications.

Figure 4-3. Print Priority Settings



Printer Object

A *printer object* represents a local or network printer. If you installed a printer during the OS/2 2.1 installation process, an icon for a printer object was placed on the Desktop. This printer object represents the configuration of hardware and software for the default printer you selected.

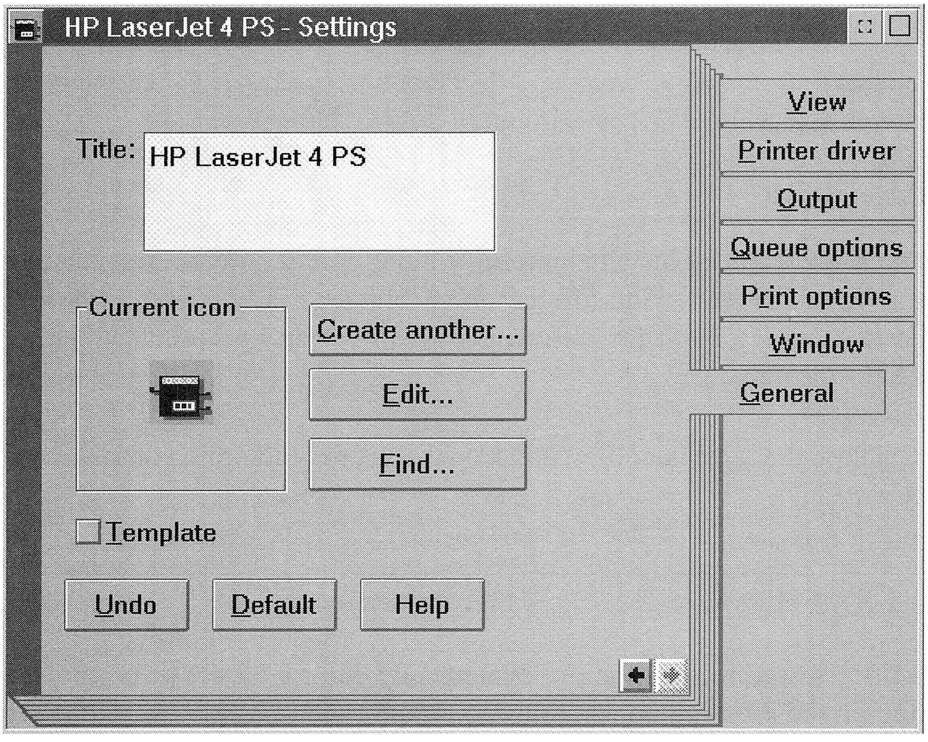
A printer object can also be created by:

- Creating a copy of an existing printer object.
- Using a template. (To create a printer object using a template, open the Templates folder, find the Printer icon, and simply drag and drop the icon to the Desktop.)

- Using the **Create another** option from the pop-up menu.

Printer objects determine where and how files will be printed. Each printer object is a print queue; OS/2 2.1 can have multiple print queues. Figure 4-4 shows the Settings notebook of a printer object.

Figure 4-4. Printer Object Settings Notebook



OS/2 removes any blanks in the name assigned to the printer object so that the same name can be used as the queue name, which is also the subdirectory name under the SPOOL directory. For example, the printer name *HP LaserJet 4 PS* would become the queue name *HPLASERJ*.

Pooling and Sharing

Multiple printers that are identically configured and use the same printer and job properties can be connected to different ports. If more than one port is selected in a printer port object, you should use *pooled printing*, which allows print jobs to be output to any selected free port. This feature is useful in a network printer server because the server dynamically directs the print job to the printer that is not busy.

The ability to assign multiple printer objects to a single printer is called *sharing*. This prevents you from having to reconfigure the print subsystem when using different printer languages or job properties. For example, sharing enables queues with different paper sizes (letter and legal) to print to the same port.

Print-Related Objects

You can access the following print-related objects through a printer object:

- Print job object
- Printer driver object
- Printer port object
- Queue driver object

Multiple printer drivers can be installed in one or more printer objects. You can also install the same printer driver (with different job properties) in two or more printer objects and switch among them for different jobs. However, one printer object is always selected as the default.

To set or change the default printer object, click on the **Set default** option of the printer object's pop-up menu.

Print Job Object

A *print job object* is used to monitor and dynamically control all the print jobs. By double-clicking on the print job object, you can monitor the print job status through either a details view or an icon view.

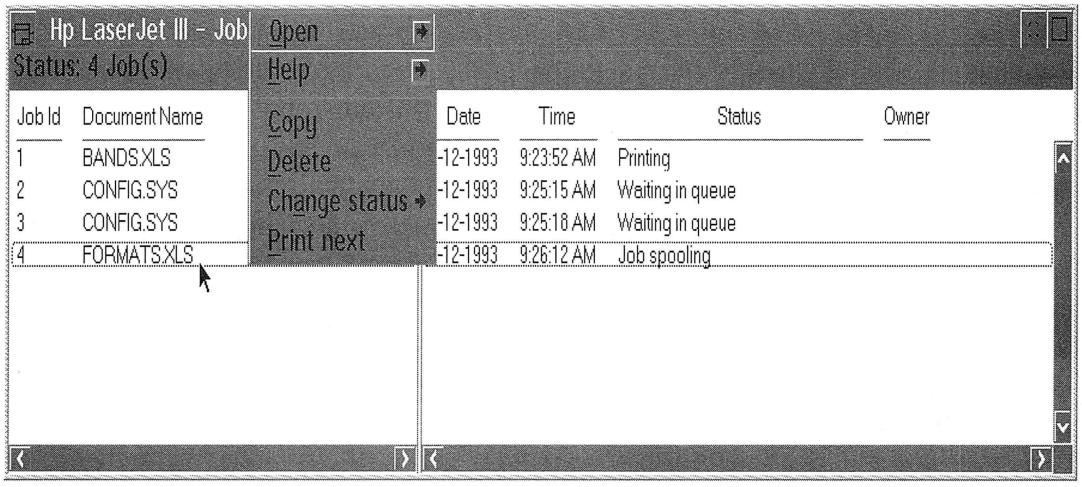
The job details view reveals a line of job information that includes job status, job contents, and any printer problems. It also includes the date, time, and owner of the print job.

The job icon view graphically displays the job status, including the names and order of the jobs. This is the default setting for a printer object.

Previewing the Print Job Status

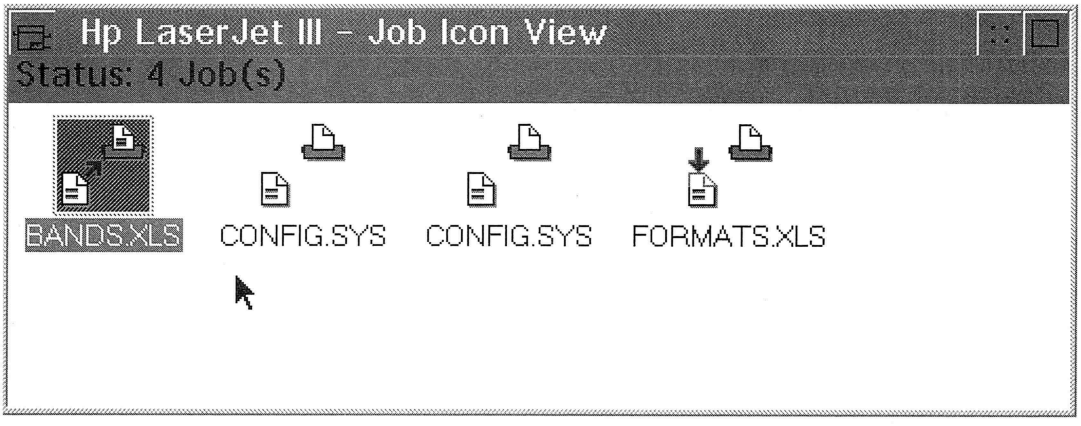
Figure 4-5 shows a details view of the job status of a printer object. It also shows the pop-up menu for the last print job. In this example, Job 1 is printing while Job 4 is spooling. You can copy, delete, or change the job priority of any job from the pop-up menu for that print job. You can also hold, release, or restart the job. The easiest way to preview a spooled job is by holding the job in the spooler queue.

Figure 4-5. Details View of a Printer Object



These print job selections also can be made through the pop-up menu of a job icon view. Figure 4-6 shows the previous job status example in an icon view.

Figure 4-6. Icon View of a Printer Object



When a job is on hold, the spooled file is held in the spooler. You can view the job size and contents and decide whether to continue printing. You can also use the metafile tool and printer language tools (for languages such as PCL and PostScript) to analyze the spooled file before printing it. This is especially helpful in large print jobs.

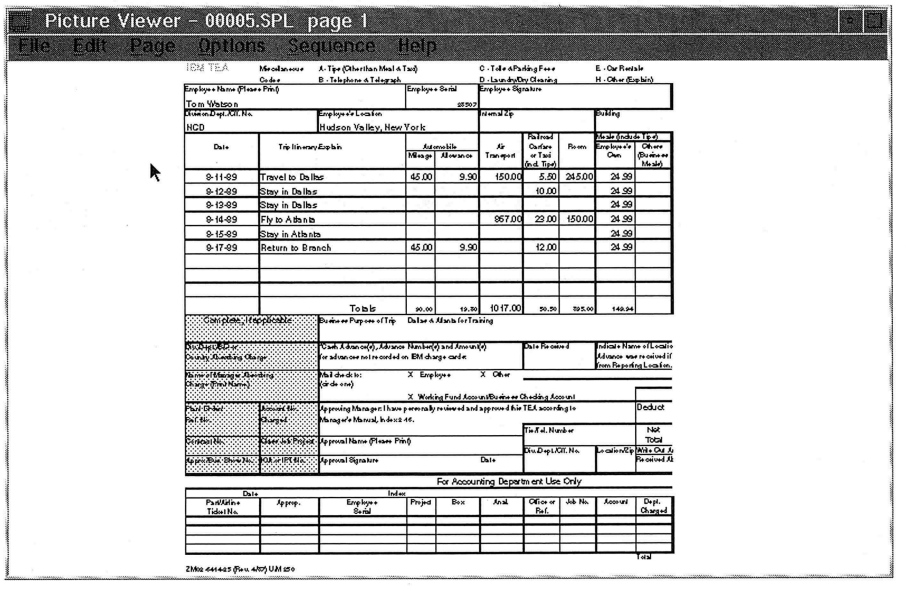
Deleting a job will stop the job from being sent to the printer. All print jobs can be stopped at any time. However, the data already in the printer buffer continues to print. A message box will appear if a job cannot be sent to the printer because of a connection problem or a problem with the printer itself. The message will prompt you to retry, abort, or cancel the print job.

Previewing the Job Contents

The contents of a spooled job can be previewed with the *Job Contents View* utility. From the print job window, double-click on a print job icon to display the job contents.

For PM_Q_STD spooled files, the Job Contents View utility invokes the metafile tool *PICVIEW*, which shows the job contents graphically. PICVIEW is used to view and print standard spooled files, multipage metafiles, and PIF files. To activate it from the command line, type **PICVIEW** to start the program. Figure 4-7 shows a job preview for an Excel application in metafile format.

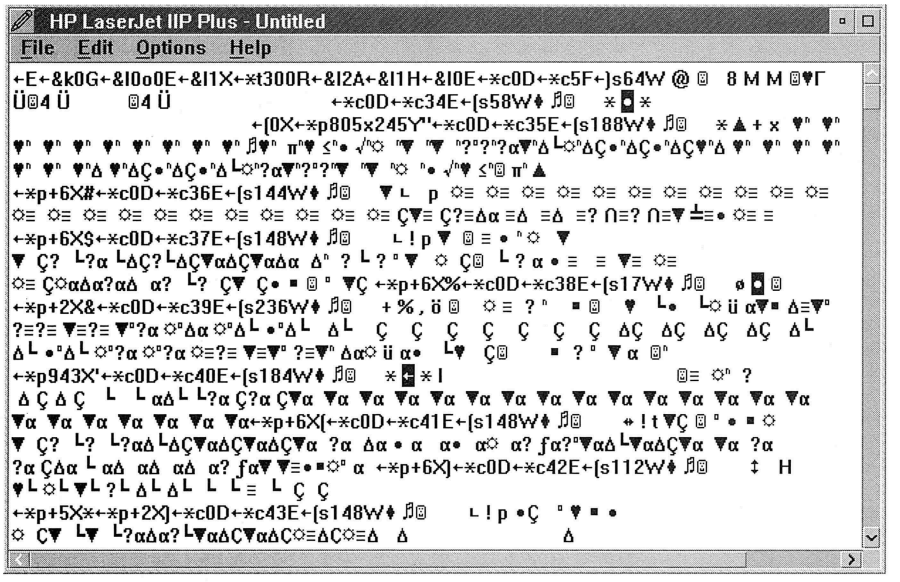
Figure 4-7. Job Preview of Excel (PM_Q_STD Format)



For PM_Q_RAW spooled files, the Job Contents View utility invokes the System Editor. This displays the printer control commands, making these files difficult to understand. If the OS/2 System Editor (E.EXE) is renamed or replaced, the raw data files preview will not work.

Figure 4-8 shows a job preview for an PageMaker application in raw data format.

Figure 4-8. Job Preview of PageMaker (PM_Q_RAW Format)

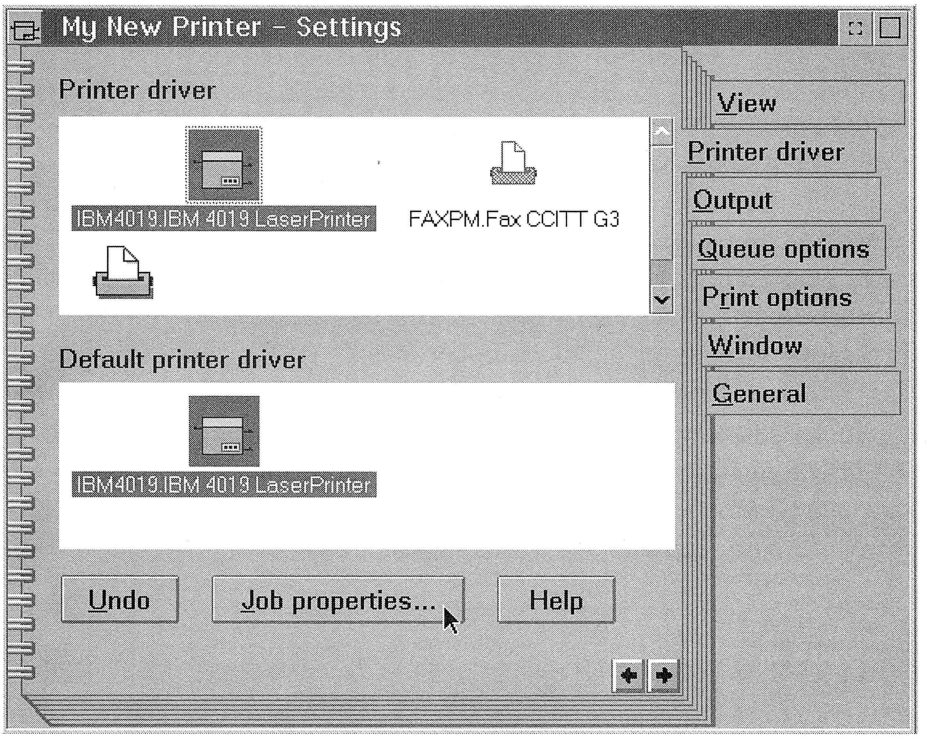


Printer Driver Object

The *printer driver object* configures the printer and sets the printer properties and job properties. To access the printer driver object, open the Settings notebook of the printer object (see Figure 4-9 on page 68). You can then install or delete printer drivers and select the default printer driver.

One printer driver can drive several printers of the same type. Conversely, one printer can be configured as several different types of printer languages and need several different types of printer drivers, one for each printer language. (See "Printer Drivers Provided with OS/2 2.1" on page 71.)

Figure 4-9. Printer Driver Object Settings Notebook



Printer Driver Extended Attributes

Each printer driver has extended attributes that include all the necessary information about the driver. Figure 4-10 shows the Extended Attribute (EA) information for the LaserJet printer driver (LASERJET.DRV).

Figure 4-10. EA Information for LASERJET.DRV

```
.ICON                = <icon>
VENDORNAME           = HP
CLASSNAME            = PCL
REQUIREDVENDORFILES  = HP_ADDF.DLL
OPTIONALVENDORFILES  = PCLHELP.HLP, HP_ADDF.SYM, HP_ADDF.MAP
REQUIREDCLASSFILES   = GENERIC.DLL
OPTIONALCLASSFILES   = *.FNT, GENERIC.SYM, GENERIC.MAP
REQUIREDDRIVERFILES  = LASERJET.DRV
OPTIONALDRIVERFILES  = LASERJET.SYM, LASERJET.MAP
.EXPAND              = Epson ActionLaser II
.VERSION             = 13.365
```

You should use only OS/2 commands to process OS/2 printer drivers. EA information cannot be copied by the DOS COPY command if the OS/2 installation procedure is

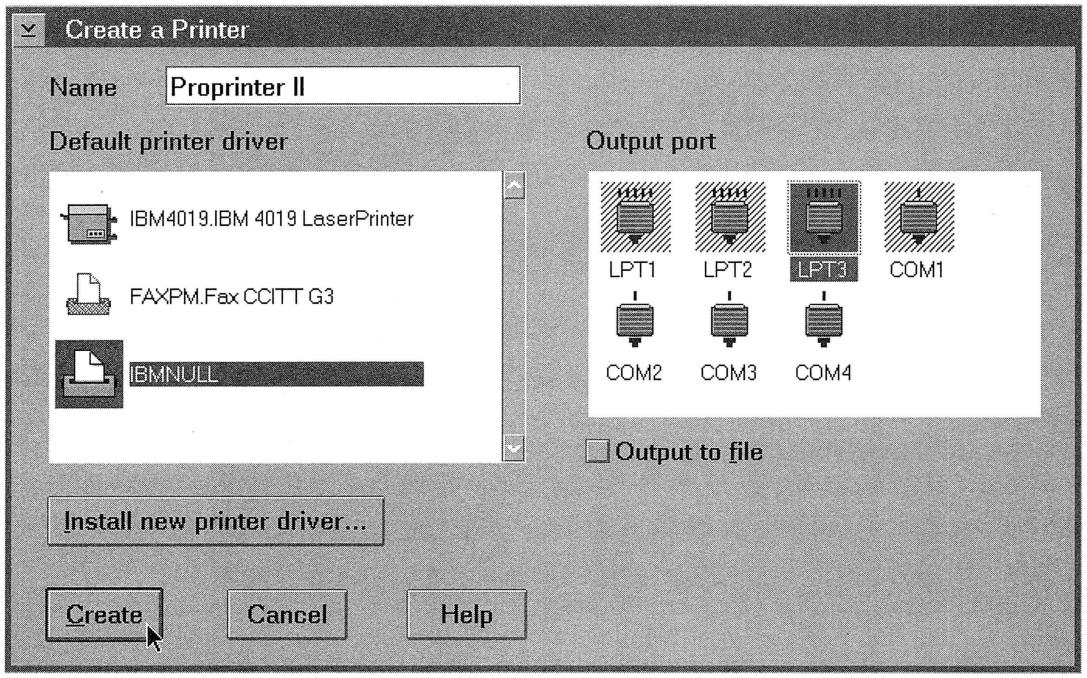
using this information to pull in the other required components. If the EA information is lost, the printer driver installation can experience problems.

Creating a New Printer Object for a Printer Driver

If you have several printer drivers installed on your system, you might want to create another printer object to use one of the drivers. To do this, drag a printer object template from the Templates folder to the Desktop or another folder. The Create a Printer window will be displayed (see Figure 4-11). Enter a name for the printer, select the port where the printer is connected, and select an existing printer driver that corresponds to your printer model. When you have made your entries, click on the **Create** button. (If you wish to install an equivalent WIN-OS/2 printer configuration, follow the instructions that will appear on the screen.)

If you need to install a new printer driver for your new printer object, follow the instructions under "Installing a New Printer Driver" on page 70.

Figure 4-11. Creating a New Printer Object



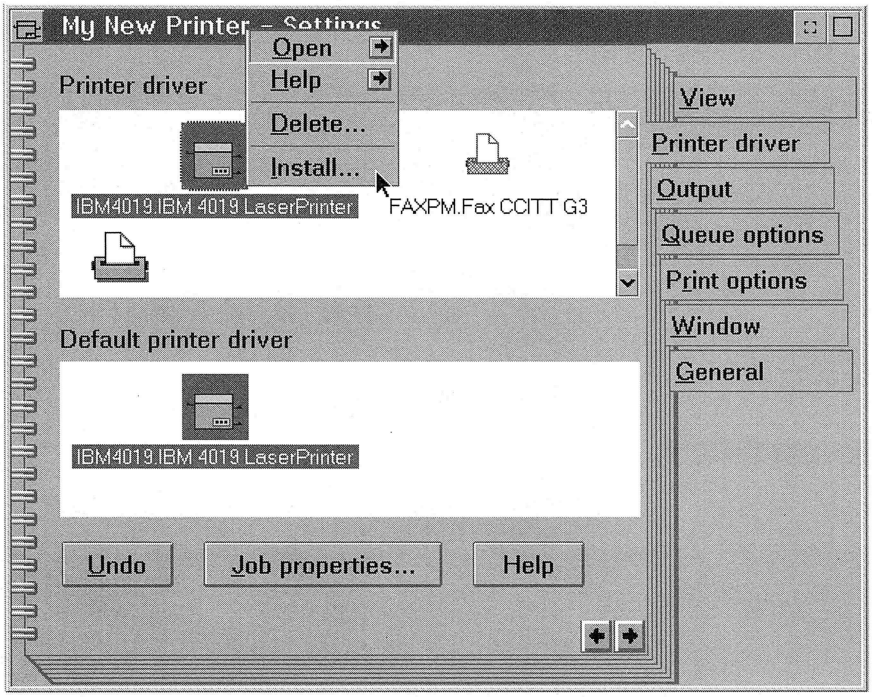
To customize the settings for your new printer object, see "Setting Printer and Job Properties" on page 73.

Installing a New Printer Driver

You can use any existing printer driver icon to install a new printer driver in the printer driver object. However, if you delete the old printer driver, be sure to use the **Delete** option from the pop-up menu for that particular printer driver icon.

To install a new printer driver in OS/2 2.1, select the **Install** option from the pop-up menu for the new printer driver (see Figure 4-12). If the new printer driver is one of the drivers provided by OS/2 2.1, select the **Printer driver shipped with OS/2** option. A list of printer drivers will appear. Select the names of the printer drivers you wish to install and click on the **Install** button.

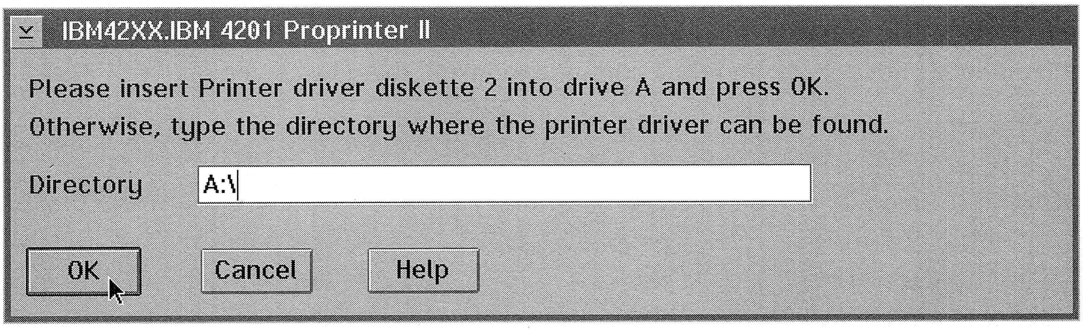
Figure 4-12. Installing a Printer Driver



To install an OEM driver or any other printer driver that is not on the system printer driver diskettes, use the **Other printer driver** option on the Install menu. A pop-up window will appear and prompt you for the diskette to insert or request the directory where the printer driver can be found (see Figure 4-13). A list will appear containing

all the names of the printer models that use this driver. You can then select the one you want.

Figure 4-13. Other Printer Driver Pop-Up Window



Each printer driver is usually responsible for a number of printers. For example, the LaserJet printer driver is responsible for the HP LaserJet 2000, 500, Classic, II, IID, IIP, III, IIID, IIIP, 4, and the OEM LaserJet-compatible printers. If you have already installed the HP LaserJet II printer driver, you can install the LaserJet III printer driver by simply referencing this file, which already exists on your hard disk.

Printer Drivers Provided with OS/2 2.1

There are 11 printer/plotter drivers in the OS/2 2.1 print subsystem (see Table 4-1). These drivers can drive 280 different printer/plotter models.

Table 4-1. OS/2 2.1 Printer and Plotter Drivers

Printer Driver	Name
IBM4019.DRV	IBM 4019 printer driver
IBM42XX.DRV	IBM 42XX printer driver
IBM52012.DRV	IBM 5201-2 printer driver
IBM52XX.DRV	IBM 52XX printer driver
IBMNULL.DRV	IBM Null printer driver
EPSON.DRV	Epson printer driver
HPDJPM.DRV	HP DeskJet printer driver
LASERJET.DRV	HP LaserJet III printer driver
PLOTTERS.DRV	Plotter driver
PSCRIPT.DRV	PostScript printer driver
SMGXPJET.DRV	PaintJet printer driver

If you get a new printer driver for a current or new printer model (such as the LaserJet 4M), be sure to delete *all* the old (in this case, LaserJet) printer model icons because they use the same driver name as the new printer driver. A message box should appear asking whether to delete the files from your disk. If you do not see this message, there are one or more printer drivers still associated with the printer icon. Make sure that all the old icons are deleted and then continue to install the new driver.

TIP

Delete all associated printer drivers before you install a new one. In the OS/2 operating system, the printer driver (.DLL and .DRV) files are locked by the system if they have been used by an application or used from the setup of that printer driver. You cannot replace or delete that printer driver until you reboot the system.

Each printer can have different configurations, that is, one printer can be configured for many different printer types. For example, the IBM 4029 printer has four emulation modes: IBM PPDS, PCL4, PostScript, and Plotter.

You should use different printer drivers to match the different configurations. It is important that you select the proper printer driver names and correctly set the printer emulation mode.

For the IBM 4029 printer, use the following printer drivers for the different emulation modes:

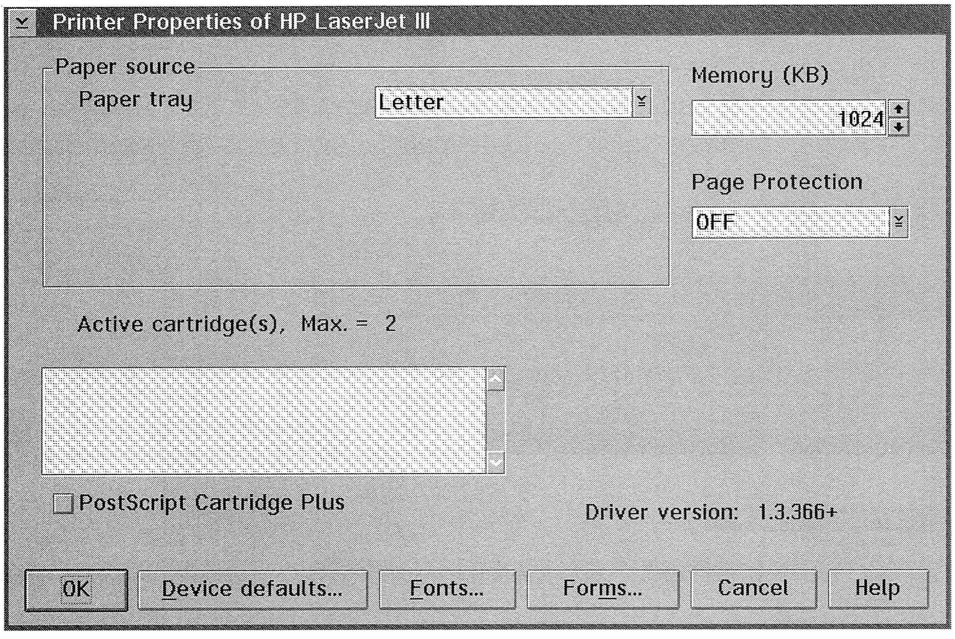
- IBM 4019.IBM4029 driver (on Printer Driver Diskette 2) for IBM 4029 PPDS emulation mode.
- LaserJet.IBM4029 driver (on Printer Driver Diskette 1) for IBM 4029 PCL4 emulation mode.
- PostScript.IBM4029 driver (on Printer Driver Diskette 1) for IBM 4029 PostScript emulation mode.
- Plotter.IBM6180 driver (on Printer Driver Diskette 3) for IBM 4029 Laser printer in Plotter mode.

An emulation mode can also be set manually on the printer. Some printers with multiple emulation modes are capable of changing their mode through the application software. Several OS/2 2.x printer drivers, such as the LaserJet driver for the IBM 4029 Laser printer, support *Set Initial Conditions (SIC)* code, which automatically switches the printer to the correct emulation mode.

Setting Printer and Job Properties

The settings for each system printer are called *printer properties* (see Figure 4-14). An application uses these settings to select the properties related to a physical printer, such as the type of paper trays, active font cartridges, page protection, and memory size in KB. To get to these settings, click on the **Open** option in the pop-up menu of the printer driver object.

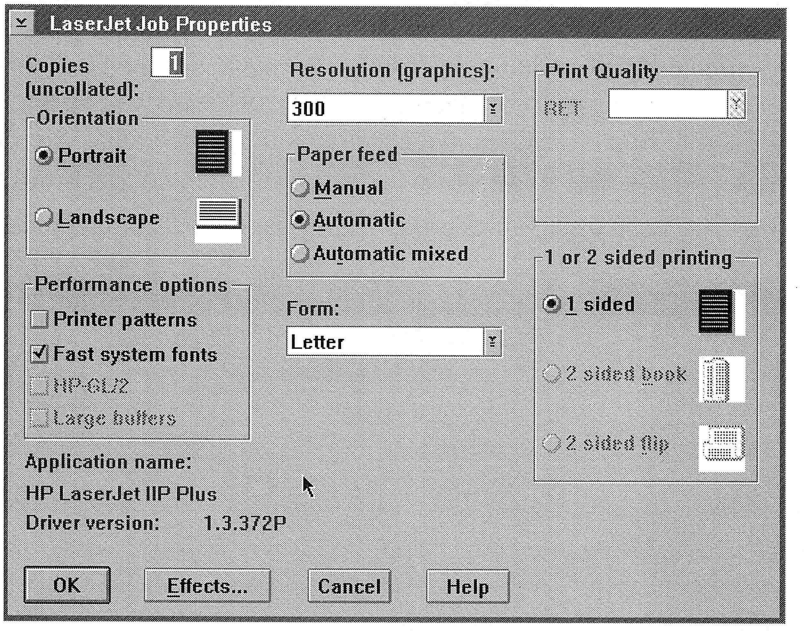
Figure 4-14. LaserJet III Printer Properties



The settings for each print job such as paper orientation, default paper tray, and printing resolution are called *job properties* (see Figure 4-15 on page 74). To get to these settings, click on the **Job properties...** button on the Printer driver page of the printer object Settings notebook.

Default job properties are stored on a per-queue basis. The settings also can be changed on a per-job basis. Because of this, setting job properties allows you the flexibility to send jobs with different settings to a single printer. It is up to the application to take advantage of this function if it has the capability; otherwise, you must set the default job properties through the printer object. You can also instruct the printer object to prompt you for the job properties and change them each time you print. (See "Setting Queue Options" on page 80.)

Figure 4-15. LaserJet III Job Properties



Some job properties may appear to overlap with printer properties. This is because the job properties setting is selectable on *each queue* while the printer properties setting is selectable on *each printer*. Most OS/2 applications use job properties for their print settings.

Deciding Which Printer and Job Properties to Use

The following questions on printer and job properties and their answers use the LaserJet printer driver as an example.

Should I select paper form or paper tray?

If there is more than one tray in your printer, you cannot directly select a default tray on the Printer Properties or Job Properties menu. This is because printer drivers select the paper tray according to the paper form defined in their Job Properties menu, not by the paper tray number or position.

For example, if a LaserJet IID printer driver has printer properties set so that the upper tray is letter size and the lower tray is legal size, the lower tray will be used automatically if you select **LEGAL** as the paper form in the Job Properties menu. If both trays are letter size, the upper tray is used as a default.

To use the lower tray as a default, you should select the paper form of the lower tray by performing the following steps:

1. Go to the Printer Properties menu and select the **Forms** option.
2. Define a new paper form name for the lower tray (for example, "Letter2") and input the paper size (the same as the letter size).
3. Select "Letter2" as your lower tray paper form.
4. Select "Letter2" as the default paper form name in the Job Properties menu. The lower tray will now be selected automatically.

Automatic mixed mode can also be used to select the paper form in a LaserJet printer driver. You can select one paper form for the cover page and another paper form for the rest of the pages. For example, to use the lower-tray paper for the first page and upper-tray paper for the rest of the pages, select the paper form of the first page to match the lower tray and select one of the remaining pages to match the upper tray. If both of them are letter size, the lower-tray paper should be defined as another name, as mentioned previously.

How do I select the printer memory size?

Usually, you should set the printer memory size in the Printer Properties menu to match the memory size you have installed in your printer. For LaserJet III printers, you should also select the page protection size to be the same as the setting on your printer. A LaserJet driver uses the printer memory size and page protection size (if any) to decide its working mode, which can be compression or noncompression mode.

What are print effects?

The **Effects** option enables you to select printer patterns for the foreground and background. The patterns can be selected from different shades (1% to 99%) or different pattern types (Pattern 1 to Pattern 6). We suggest that you select a pattern for the background (for example, 2% shade) to generate output with a slightly shaded background. In many cases, this makes the output look better than the regular paper (white) background.

How do I install a cartridge or soft fonts?

The list in the Active Cartridge box in the Printer Properties menu contains the names of the currently available cartridges and software fonts (soft fonts). You can select one of these cartridges or soft fonts, or you can install new ones.

To install a new cartridge font, use the **Fonts** option in the Printer Properties menu to access the font installer window, insert the font description diskette (which comes with the cartridge), and add the new font to the list.

To install a new soft font, use the **Fonts** option again, type the name of the soft font directory, click on the **Open** button, and select the fonts you want to install.

Print Performance Options

The following print performance options are added to the OS/2 2.1 printer drivers for LaserJet and IBM 4019 printers:

- Fast system fonts (LaserJet/IBM 4019)
- Printer patterns (LaserJet)
- HP-GL/2 (LaserJet)
- Large buffers (LaserJet)

Generally, the printer performance options are used to decrease the size of spooled files and print more quickly. However, there might be some limitations with some applications, so use these options cautiously.

Fast System Fonts

This option improves printer performance in LaserJet, IBM 4019, and some other printer drivers by downloading the system bit-map fonts to printer memory. The system fonts are then treated as device fonts. This allows the job to print faster and requires a smaller spooled file. Print speed is dramatically improved in large text files (such as spreadsheets) that are designed in OS/2 system fonts. However, because the system fonts are no longer treated as graphics, some OS/2 graphics abilities can be lost when using these fonts.

If the **Fast system fonts** option is disabled, the system fonts do not need to be completely printed out like device fonts; instead, the OS/2 graphics system can partially clip them. Also, the OS/2 graphics system can handle system fonts that overlay graphics in opaque or transparent mode; device fonts can only do this for simple cases. Because of these differences, you should carefully consider your needs before enabling this option. You should disable this option if the printed output is different from what is shown on the screen.

Printer Patterns

This option can be added to LaserJet and some other printer drivers to enable you to send pattern print commands directly to the printer instead of sending the OS/2 graphics engine print commands.

Pattern drawing is widely used in graphics drawing to fill the areas and paths. Many complicated drawings are combined by large numbers of small pattern filling. Using the **Printer patterns** option enables graphics jobs that contain shaded and patterned rectangular areas to print faster and keep smaller spooled files. However, the printer pattern will not exactly match the screen pattern, and this option will not work well with large pattern numbers.

The **Printer patterns** option should be turned off when using graphics that overlap with patterns.

HP-GL/2

This option is a new feature in the Printer Control Language 5 (PCL5) that speeds up graphic print jobs by enabling more graphic commands to be directly executed by the printer. This option is added to the LaserJet printer drivers.

> **TIP**
>
> When using HP-GL/2 in complicated graphics cases, the printer may report a Printer Overrun Error (Error 21). To prevent this problem, turn this option off or turn Page Protection on at the printer.

Large Buffers

This option is another new feature of OS/2 2.1. Because it requires a lot of memory, it is not generally used. If you have enough free memory, however, checking this option will enhance performance by printing your output more quickly.

Printer Port Object

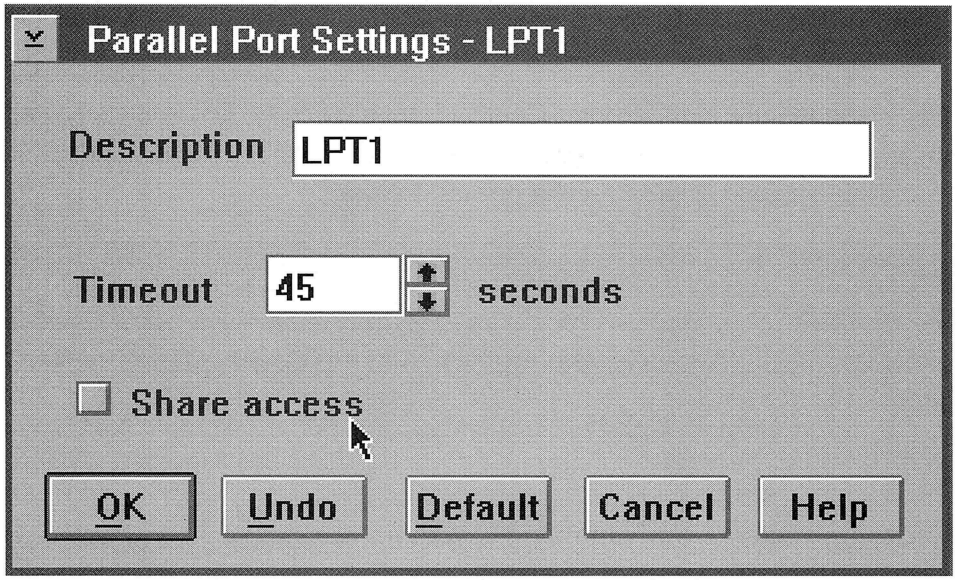
The *printer port object* allows the user to select the output port and set the port transmission parameters. From the **Output** option of the Settings notebook for a printer object, you can select the output port or redirect the print data to a file. If

Output to file is selected, a message box will prompt you to input the output file name when a print job is started. The output file contains the print-specific data that is to be sent to the printer. This data can be copied to, and printed from, any machine (running any operating system) that is set up for the specified printer. **Output to file** is used only for OS/2 applications.

Selecting an Output Port

A parallel port or serial port can be selected as the output port. For parallel ports, a *timeout value* can be set (see Figure 4-16). This value equals the number of seconds that the computer tries to print before it determines that there is a print-error condition and changes the printer object status to Error. The default value is *45*, which is appropriate for most printers.

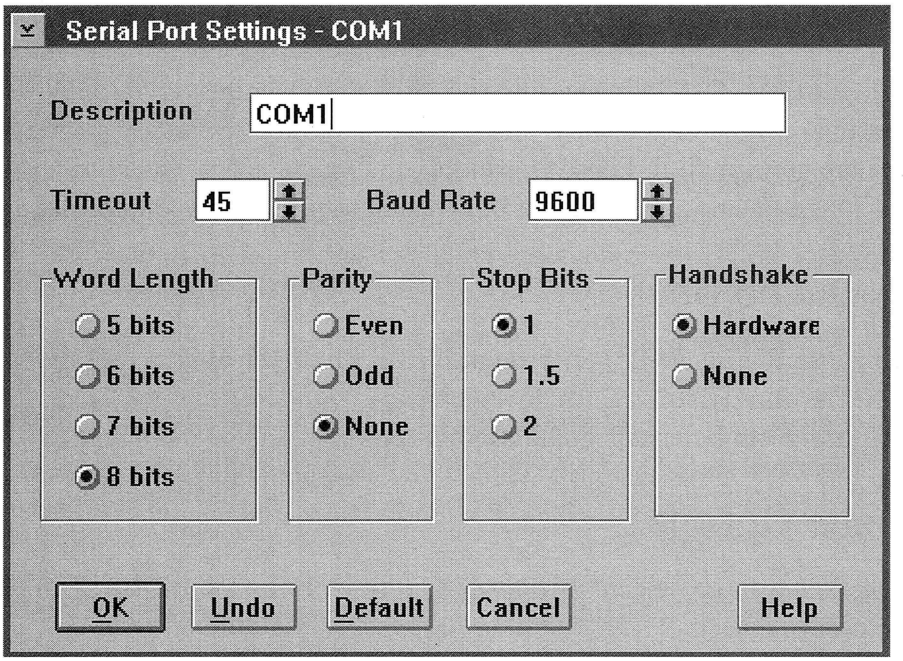
Figure 4-16. Parallel Port Settings



The **Share access** option allows DOS security device applications to access the parallel port hardware simultaneously with the system. DOS applications can query the security device for copyright protection. If an application tries to access a port during initialization, an error will occur if the port is already in use. Checking the **Share access** option prevents this type of error.

For serial ports, you should set all the serial communication parameters to match the serial printer (see Figure 4-17). These parameters include baud rate, word length, parity, stop bits, and handshake mode.

Figure 4-17. Serial Port Settings



In addition to selecting the system port settings, you should also manually select the port settings on the printer; some printers have special procedures for this. You can find this information in your printer's documentation.

Parallel Port and Serial Port Device Drivers

Device drivers, also known as *kernel* or *physical* device drivers, are the interface between the operating system and the ports to the physical printers.

The following list contains examples of device drivers:

PRINT01.SYS For parallel ports of ISA and EISA machines.
PRINT02.SYS For parallel ports of Micro Channel machines.
COM.SYS For serial ports on all machines.

When loading a parallel port device driver, the appropriate statements are automatically added to the CONFIG.SYS file. However, statements for a serial device driver must be specified in the CONFIG.SYS file during its installation.

Queue Driver Object

If the spooled file is in PM_Q_STD (metafile) format, it must be replayed through the Presentation Manager printer driver when it is sent to the printer. In Phase I printing, the spooled files are generated. In Phase II printing, the *queue driver object* submits the spooled files to the PM printer driver for conversion to printer-specific format.

Installing a Queue Driver

You can select or install a queue driver through the queue driver object. (Normally, *PMPRINT* is used as the default queue driver.) If *reverse clipping* is required, the PMPLOT queue driver can be installed and used instead.

Setting Queue Options

The following options can be selected from the **Queue options** tab of the Settings notebook of the printer object:

- **Job dialog before printing**. Select this checkbox to specify job properties on a per-job basis, instead of using the default job properties set in the printer driver object.If this option is checked, then whenever you print using drag and drop or select the **Print** option from a WPS object pop-up menu, a Job Properties window for that particular job appears.
- **Printer specific format**. Selecting this checkbox generates all the spooled files in PM_Q_RAW format (printer-specific format). This format takes more storage space than the printer-independent format. It is primarily used for networking to servers that do not support OS/2 Presentation Manager.
- **Printing while spooling**. This is the default. This checkbox starts the printing thread while the spooling thread is still receiving data. This can speed up printing speed, especially in multipage documents, because the first page starts printing as soon as the raw data file for this page is generated.

OS/2 Print Utilities

There are many methods of printing in OS/2 2.1; for example, you can perform any of the following actions to print in OS/2:

- Use the PRINT or COPY command from the command line prompt.
- Drag and drop a data file object onto a printer object.
- Use the print screen function key.
- Instruct the application program to print.

Using Command Line Printing

At the command line prompt, you can use the PRINT or COPY command to print a data file. The PRINT command sends a data file to a print device (the default device is LPT1 if a print device is not specified). You can also use the /D:DEVICE option to specify the device (LPT1 - LPT3 for regular use and LPT4 - LPT9 for printing to a network printer). The PRINT command prints an ASCII file to a device whose default includes end-of-file character (Ctrl+Z) processing. Ctrl+Z characters are recognized as end-of-file indicators for most ASCII files. Use the **/B** option to turn off the Ctrl+Z processing.

The COPY command sends a data file to a print device and all the other devices in the system. If you use a serial printer, this command *must* be used. Use the **/B** option to specify binary file printing or the **/A** option to turn on Ctrl+Z processing.

Text file printing should use Ctrl+Z processing; file printing generated by the **Print to file** option should use binary file printing.

ASCII files cannot be directly sent to a PostScript printer because the printer understands only PostScript language. Therefore, do not use PRINT or COPY commands to print to PostScript printers; instead, use drag and drop to print a text file to a PostScript printer.

The default device font for the printer selected as the system default is used to print an ASCII file when either the PRINT or COPY command is issued. You can change the output font by changing the font setting on the printer, if allowed.

Using Drag and Drop Printing

You can print directly by dragging your data file object to the printer object on the Desktop. However, only certain types of data files can be dragged and dropped for printing. The OS/2 print subsystem can print plain-text files, printer-specific files, and metafiles; all other file types are treated as plain text or printer specific.

By opening the settings of a data file, you can assign one or more file types to that data file. For example:

- For *plain text* types, the OS/2 operating system converts the data to device-independent text strings that are then converted to printer-specific format and printed.
- For *printer-specific* types, drag and drop is used to directly send the data file "as is" to the printer device.
- For *metafile* types, PICVIEW is activated to read the metafile format and convert it into the printer-specific format.

Plain text is the default type of a data file. The system always displays a message box to ask if the data file type is plain text or printer specific. If plain text is defined for a text file, the font used is defined by the system. If the printer-specific format is defined, the printer default font is used.

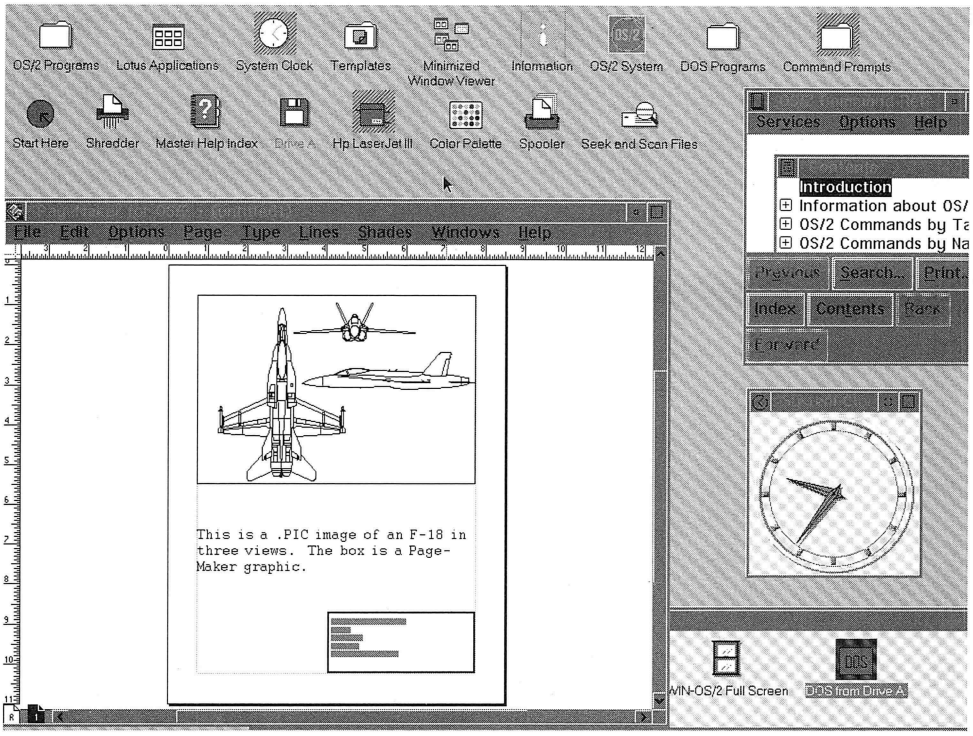
Using the Print Screen Key

By pressing the **Print Screen** function key, any window or the entire Desktop can be printed. Print Screen is enabled by default. It can be disabled by changing the settings of the **Print screen** tab in the Settings notebook of the system object in the System Setup folder.

Print Screen takes a little longer to print than regular printer jobs because it has to get the screen bit map and convert it into the printer-specific format. To print the full screen, place the pointer on the Desktop away from any open windows and press the **Print Screen** key.

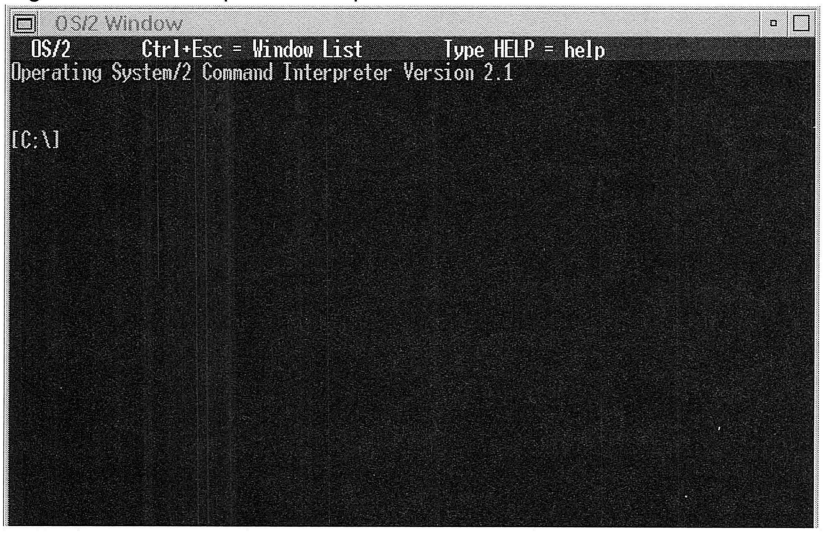
Figure 4-18 on page 83 shows a full-screen printout.

Figure 4-18. Example of a Full-Screen Printout



To print an opened window, place the pointer in the window you want to print and press the **Print Screen** key. Figure 4-19 is an example of an opened window printout.

Figure 4-19. Example of an Opened Window Printout



Part II. Designing OS/2 2.1 Applications

Chapter 5.
Understanding Common User Access Interfaces

The *Common User Access (CUA)* component of IBM's Systems Application Architecture (SAA) is a graphical user interface that defines a consistency of appearance, operation, and terminology across applications that are to be used in object-oriented environments such as OS/2 2.1. Consistency across applications provides many benefits: applications can be created using a *building-block* format, which reduces compatibility errors and programming time; users can learn to use an application without having to understand the mechanisms that make the application operate; and users need less training time to effectively transfer their ability to use one application to multiple applications.

A set of design principals called the *CUA guidelines* supports the development of OS/2 applications by providing specific rules on how information should be conveyed to the end user. These rules are based on a *users' conceptual model* in which users gain an understanding of their (in this case, computer) environment by using a set of procedures that provide an expected result. Once these procedures become familiar, users can predict that using these procedures in a similar situation will create similar results. For example, if users learn that double-clicking on a particular icon (a picture representation of an application) will start or *open* that application, they will have a tendency to assume that double-clicking on another icon will open another application.

This process promotes the users' confidence in their ability, increases their productivity, and decreases the likelihood of errors. In simpler terms, this means that a user should be able to intuitively figure out the "What do I do now?" problems without having to call for technical support.

CUA and the Workplace Shell

The terms and techniques outlined in this chapter should give you a basic understanding of how the Workplace Shell implements the CUA guidelines. For detailed information on the CUA architecture, refer to IBM's *Common User Access Advanced Interface Design Reference* in the OS/2 2.0 Technical Library.

NOTE

Some differences exist between the Workplace Shell and CUA that might sometimes require you to diverge from complying with the CUA guidelines. If this occurs, use the OS/2 conventions to ensure system consistency.

The CUA guidelines operate on several levels of application design:

- Implementation of the object-oriented model.
- Application of similar behavior to all user interface controls through use of the same keys and terms, such as F1 for Help.
- Presentation of consistent visual representations.

Implementing the Object-Oriented Model

The *object-oriented* model simplifies the task performance for a user by enabling the user to focus (orientate) on an object and the actions related to that object. Implementation of this model requires an understanding of objects, object classes, and object hierarchy and inheritance. These terms and concepts are described fully in Chapter 6, "Understanding the System Object Model."

Types of Objects

The CUA guidelines define three types of objects: container, data, and device.

Container Object

A *container object* contains other objects. It enables a user to group related objects into one object (see Figure 5-1). The user can easily access and retrieve information from these objects.

Data Object

A *data object* can contain various types of information such as audio, video, text, or graphics (see Figure 5-2).

Device Object

A *device object* is generally a representation of an actual *physical* object. For example, a printer is represented by a printer (device) object (see Figure 5-3 on page 90). Device objects can also represent non-physical or *logical* objects in the system. For example,

Figure 5-1. Example of a Container Object

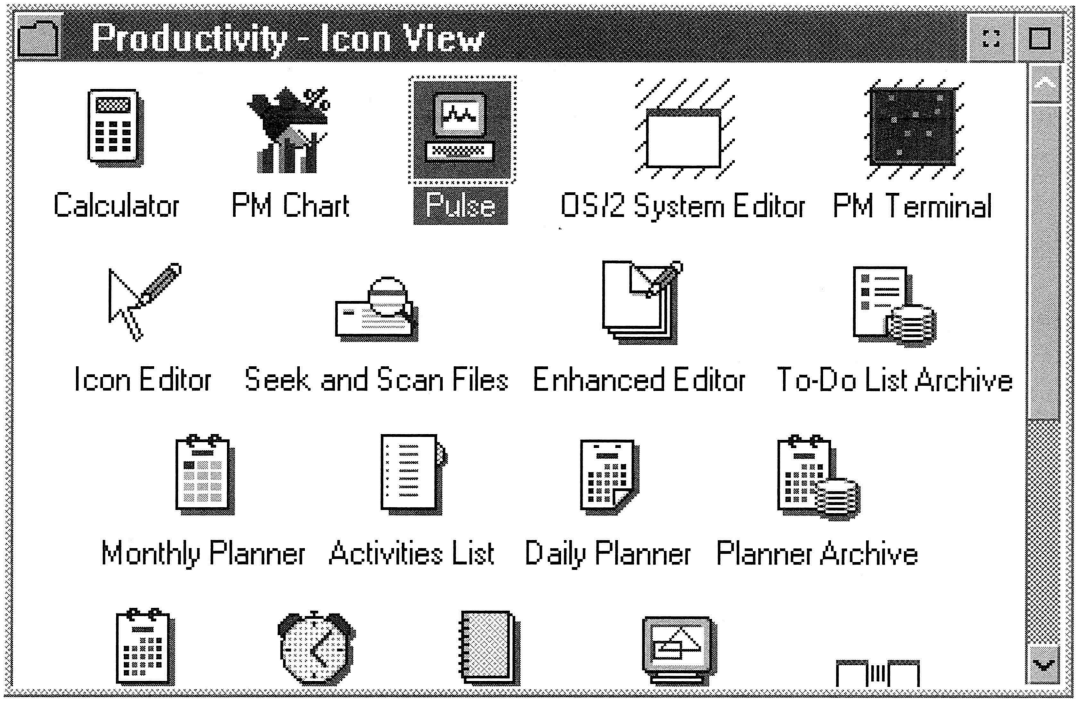
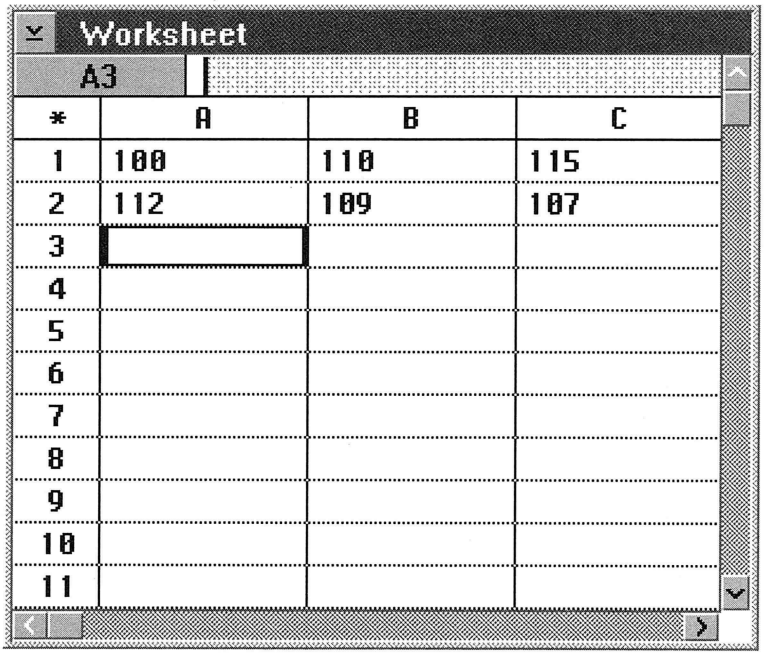


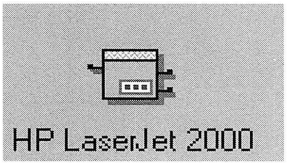
Figure 5-2. Example of a Data Object



*	A	B	C
1	100	110	115
2	112	109	107
3			
4			
5			
6			
7			
8			
9			
10			
11			

a shredder object is an example of a logical object that disposes of other objects or data.

Figure 5-3. Example of a Printer (Device) Object



Some device objects are also container objects. For example, a printer object might contain a list, or *queue*, of objects to be printed.

Viewing an Object

A user accesses an object by opening a window to view the contents of that object. Each *view* is a different way of looking at the object and its information. An object can have more than one view. There are four basic view types:

- Settings view
- Composed view
- Contents view
- Help view

Settings View

A *settings view* displays the attributes of an object and is generally provided for each object type (see Figure 5-4). In this view, the user can change the attribute settings of the object, if desired. However, some settings such as the object's creation date, cannot be changed by the user.

Composed View

A *composed view* displays an object's data in a meaningful order (see Figure 5-5). If that data is arranged in a different order, the object will have a different meaning. For example, graphs are usually displayed in a composed view because the arrangement of the elements in the graph determines the meaning of the graph. If the arrangement is changed, the representation of the object's data is changed; therefore, the meaning of the graph (object) also is changed.

Figure 5-4. Example of a Settings View

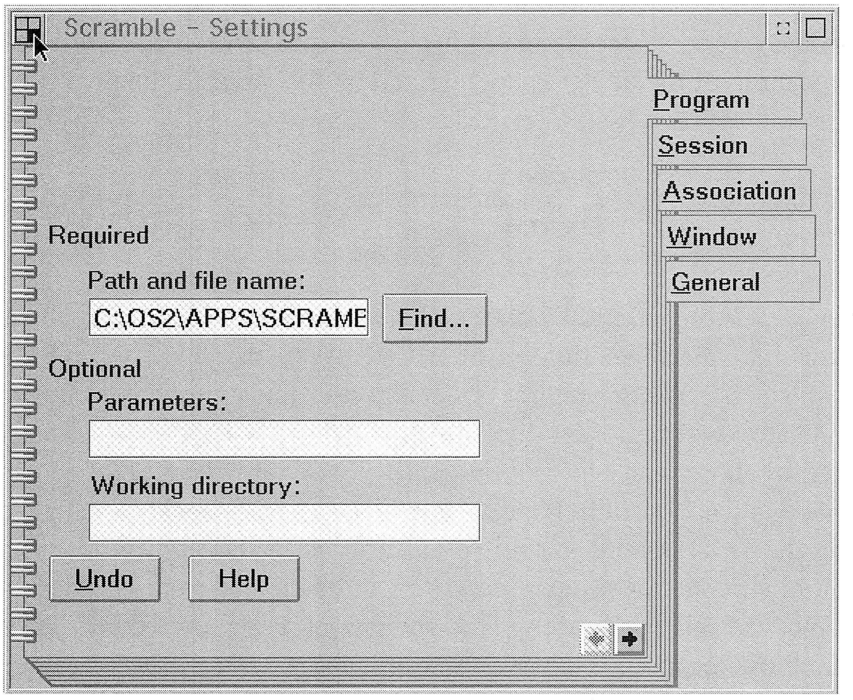
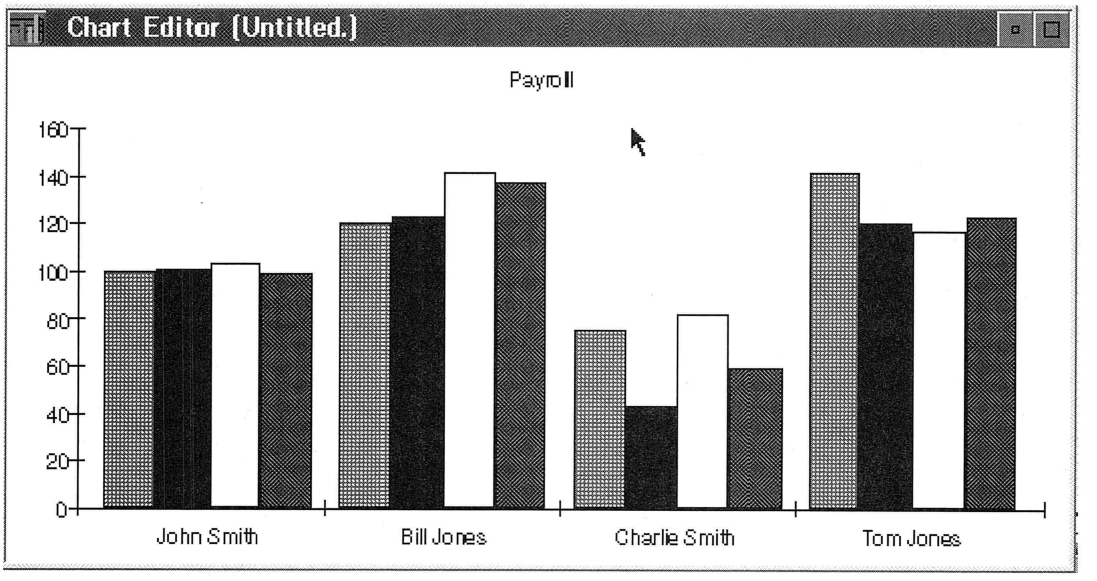


Figure 5-5. Example of a Composed View



Contents View

A *contents view* displays an object's components in a list format, which can be ordered or unordered. The meaning of the object is not affected by the order of the displayed components.

There are two kinds of contents views: icon and details.

Icon View:

An *icon view* displays each object as a symbol called an icon (see Figure 5-6). This enables a user to manipulate or position an object easily. Conveying the object's purpose graphically also reduces the amount of detail a user must know to complete a task. Generally, one icon will represent one object. However, it is possible to have an object represented by more than one icon, especially when it is desired to access that object from more than one location. For example, if a user has a device icon (such as a fax/modem) on the Desktop and desires to have access to this object from an application, an additional icon, known as a *shadow*, can be created to represent that same fax/modem object in the application. This shadow object allows the original object to reside in multiple locations. Unlike a copy of an object, data that is changed in the shadow object will also be changed in the original object. For more information on object shadows, see Chapter 7, "Using the Workplace Shell Environment."

Figure 5-6. Example of an Icon View



Details View:

A *details view* provides additional information about objects by displaying icons that have accompanying descriptive text (see Figure 5-7). The type of object (and the tasks that can be performed with it) determine what information is to be displayed. This view enables a user to access information about an object without having to actually open the object. This means that the user is provided a "peek" at some of the object's information.

The icons used in a details view are usually smaller than the icons on the Desktop, but they perform in the same manner; that is, they allow the user to quickly recognize and manipulate objects.

Figure 5-7. Example of a Details View

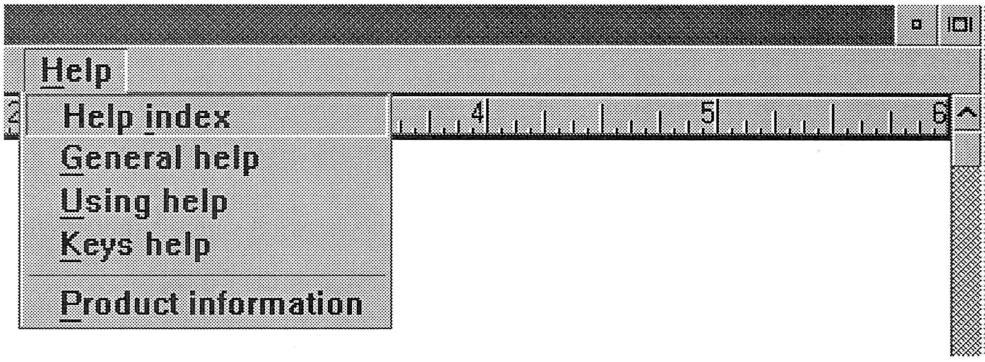
OS2 - Details View

Icon	Title	Real name	Size	Last write date	Last write time	Last ac
	APPS	APPS	0	3-19-93	2:02:26 PM	
	ICONEDIT.EXE	ICONEDIT.EXE	157,168	3-15-93	10:32:30 PM	
	BITMAP	BITMAP	0	3-19-93	2:02:26 PM	
	BOOK	BOOK	0	3-19-93	2:02:26 PM	
	DLL	DLL	0	3-19-93	2:02:24 PM	
	DRIVERS	DRIVERS	0	3-19-93	2:02:26 PM	
	HELP	HELP	0	3-19-93	2:02:24 PM	
	INSTALL	INSTALL	0	3-19-93	2:02:24 PM	
	MDOS	MDOS	0	3-19-93	2:02:26 PM	
	SYSTEM	SYSTEM	0	3-19-93	2:02:26 PM	
	000000.BIO	000000.BIO	2,560	3-12-93	5:57:22 PM	
	ABIOS.SYS	ABIOS.SYS	103	3-19-93	2:10:56 PM	
	ANSI.EXE	ANSI.EXE	5,139	3-16-93	1:49:52 AM	
	ATTRIB.EXE	ATTRIB.EXE	36,765	3-16-93	1:58:02 AM	
	BACKUP.EXE	BACKUP.EXE	28,902	3-16-93	2:00:36 AM	

Help View:

A *help view* displays information to assist a user in working with an object (see Figure 5-8). The type of information displayed is dependent upon the type of help requested by the user. For example, the user can request general help or context-sensitive help. See Chapter 10, "Creating Effective Help for Your Application."

Figure 5-8. Example of a Help View



Applying Similar Behavior to Interface Controls

The CUA guidelines define the expected behavior of methods or *interface controls* when they are selected by the user. For example, there are several ways to select an object: you can use the arrow keys and Home, End, PgUp, and PgDn keys, a specially

designated function key, or a mouse. Emphasis is placed on the manipulation of objects through their icon symbols on the OS/2 Desktop. For example, a printer would be represented on the Desktop by an icon that looked like a small printer. By moving a file object to a printer icon, the file is printed (manipulated).

In OS/2 2.1, users interact with objects to perform tasks. This interaction is accomplished by either indirect or direct manipulation.

Using Indirect Manipulation

A user generally selects and opens an object by double-clicking the mouse on an icon. Once the object is opened, the system can display the actions that can be applied to that object in a pop-up menu. The user then selects the desired action. Some actions contain multiple choices, requiring further selection by the user. This type of interaction with an object is called the *object-action* model or *indirect manipulation*. For more information on pop-up windows, see Chapter 7.

A set of actions can be performed on any object; different objects can have different sets of actions. The action sets or choices depend upon the contents of an object, its current state, and position.

Using Direct Manipulation

Users also can interact with an object by using a pointing device such as a mouse to *directly manipulate* the object. This is similar to the way a user works with objects in the real world. For example, in the physical world, a user can pick up an object or file and place it in a folder. In OS/2 2.1, the user moves the object by *dragging* it to the right place and *dropping* it into the folder.

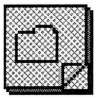
This *drag and drop* feature involves a source object and a target object; that is, the object the user is working with (source) and the object to which the source object is moved (target). The results of a "drag and drop" will vary, depending on the source and target objects. For example, if a file object is dragged to a printer icon, the file will be printed and returned to its original folder. However, if a file object is dragged to a shredder icon, the file will be deleted.

Presenting Consistent Visual Representations

The visual appearance of an application can strongly affect a user's satisfaction with your product. Therefore, it is important that the use of color, space, size, proximity, shape, etc. be carefully and consistently applied to produce an attractive and efficient interface.

To ensure that the visual interface is as familiar as possible to the user, the CUA guidelines recommend the use of visual metaphors. A *metaphor* is a means of transferring conceptual information about one object to another. For example, most users are familiar with the concept of using folders to contain files of information. By using a folder icon (see Figure 5-9) as a visual metaphor of an actual folder, the concept of a folder icon as a container of information files is easily transferred.

Figure 5-9. Folder Icon



Types of CUA Windows

A *window* is a viewport used to display data objects. A *standard* window (see Figure 5-10 on page 96) should be both sizable (to a minimized icon view or a maximized full-screen window) and movable. In addition, a standard window should contain the items shown in Table 5-1 on page 96.

A window that does not fit this criteria is known as an *optimized window* (see Figure 5-11 on page 97). These windows are used for special or specific purposes.

Windows can be primary or secondary. A *primary* window is not dependent upon any other window or object. A *secondary* window is similar in appearance to a primary window. However, a secondary window cannot stand alone; it must always be associated with a primary window and its contents are always dependent upon the object in the associated primary window. This is known as a *parent/child* relationship.

Figure 5-10. Standard Window

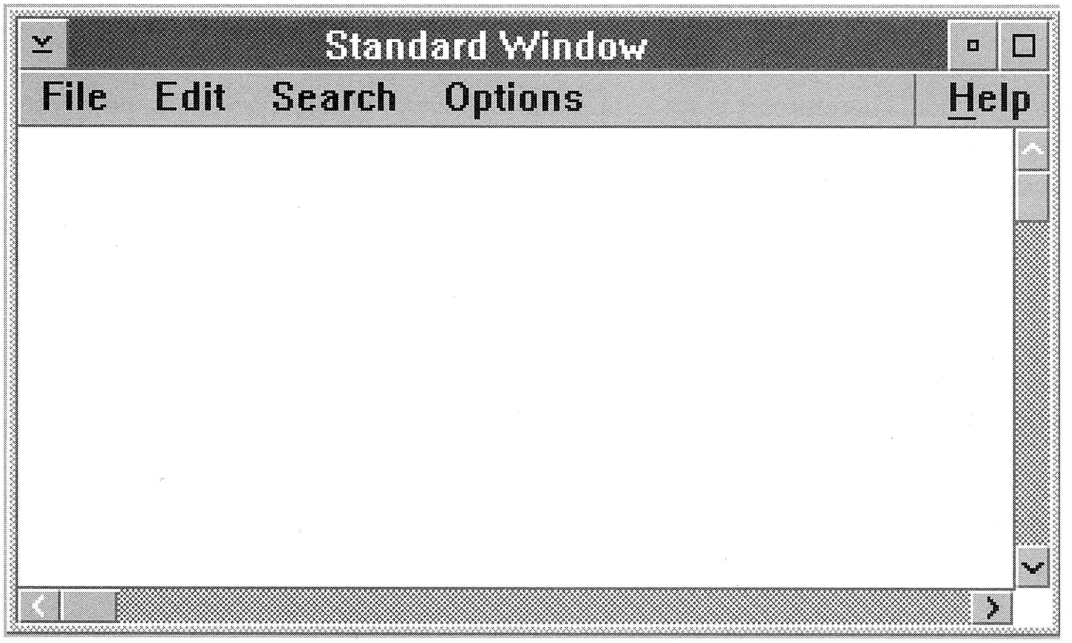
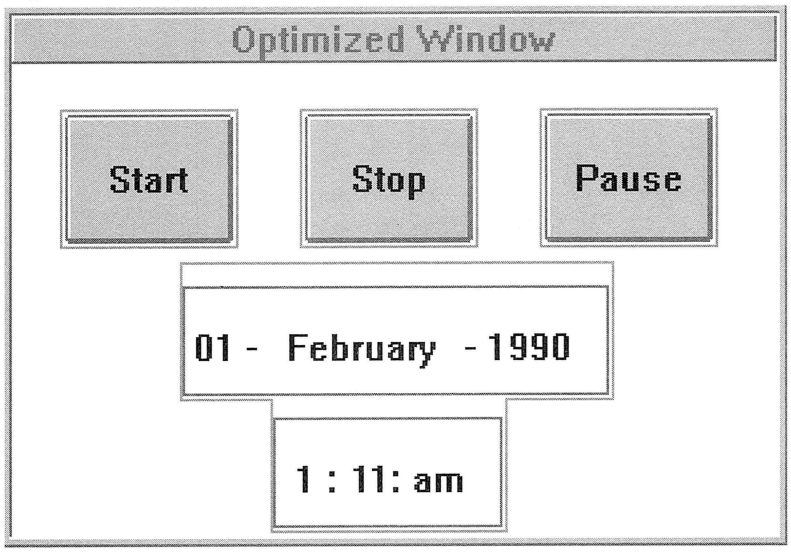


Table 5-1. Standard Window Items

Item	Description
Window frame	A line surrounding the window that separates it from other windows on the screen. Can be sized according to user preference.
System Menu symbol	Double-clicking on this symbol closes the window and ends its operation.
Title-bar icon	An icon representing the window.
Window title bar	Describes the contents of a window and provides a handle for moving the window on the screen.
Menu bar	Lists the available options. Clicking on these options will either perform an action or open another menu of options.
Minimize/maximize buttons	Minimizes or maximizes the window.
Status area	An area at the top of the screen that informs the user of pertinent data regarding the application.
Horizontal scroll bar	Allows information to be scrolled (moved) horizontally so that all information to the right or left of the window can be viewed.
Vertical scroll bar	Allows information to be scrolled vertically so that all information above and below the window can be viewed.

Figure 5-11. Optimized Window



Additional Visual CUA Components

A CUA user interface has many visual components. Some of the main components of this type of interface are listed in Table 5-2.

Table 5-2. CUA Visual Components

Component	Description
Accelerator	A *shortcut* key or combination of keys provided for frequently used menu selections. Can save time by bypassing the keystrokes needed to display the menu.
Cursor	A visual cue that indicates where the user is positioned on the screen. Cursors are moved by using a mouse (see "Pointer") or certain keys on the keyboard (Home, End, Page Up, Page Down, and arrow keys).
Folder	A general purpose container object.
Icon	A symbolic, pictorial representation of an object.
Mnemonics	An underlined character in a menu selection; when typed by the user, it quickly moves the cursor to that selection.
Pointer	A cursor associated with the pointing device of the user, which is usually a two-button mouse.
Work area	A container object in which other objects are organized according to user-specified tasks.

Chapter 6.
Understanding the System Object Model

The *System Object Model (SOM)*, which is included with OS/2 2.1, is a language-independent, object-oriented programming mechanism specifically designed for use with either procedural (nonobject-oriented) or nonprocedural (object-oriented) languages. The advantage of this is that objects written in one language can be dynamically accessed from other languages through SOM. SOM can be mastered quickly without learning a new language syntax, providing you understand the basic concepts of object-oriented programming (OOP) (see Chapter 5, "Understanding Common User Access Interfaces" and Chapter 7, "Using the Workplace Shell Environment").

As of this printing, the current release of SOM supports only the C language.

SOM consists of a run-time library and a set of utility programs that enable the creation and manipulation of objects in the system. An *object* consists of a data structure (instance variable) and the functions (instance methods) that act upon that data structure. Each object is an instance of a particular *object class*, which defines the basic elements (characteristics) of the instance variable and the parameters and implementation of its instance methods. SOM establishes a basic *inheritance hierarchy* for objects by defining and manipulating the object classes that control the relationships between objects.

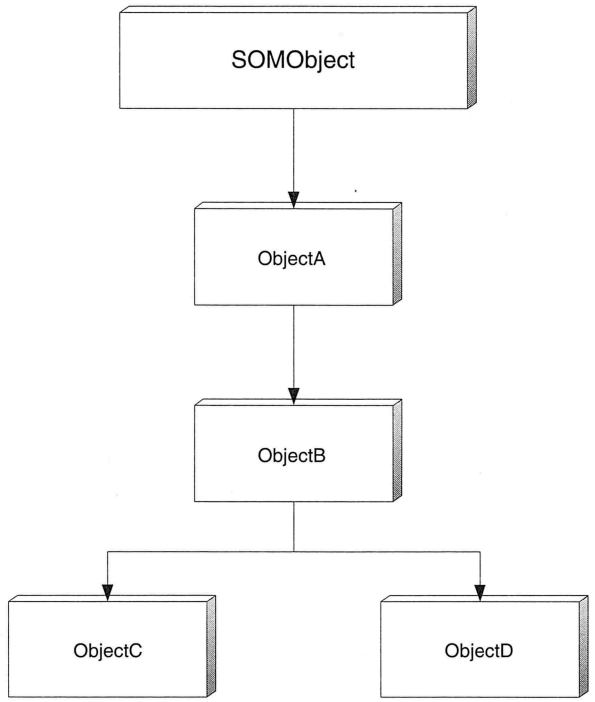
Defining Object Class Relationships

Every object class descends from a *parent class*, which in turn descends from its parent class, and so on. These parent classes are known as the *ancestors* of the object class. An object class, or *descendant*, can inherit the characteristics and behavior, including any class methods, of its ancestors. A *class method* (also known as a *factory method* or *constructor method*) is a function or routine that pertains to the object class; it can be used to create object instances.

An *instance method* is defined in an object's class definition and operates only for a particular instance of the class. (Note that an object instance must already exist for the instance method to be defined and usable.)

This inheritance relationship between objects is hierarchical. For example, if Object C is lower in the inheritance hierarchy than Objects A and B, it can contain (inherit) all the characteristics of Objects A and B and also have new and unique characteristics of its own (see Figure 6-1). Therefore, creating a new object class becomes a simple matter of *subclassing* the parent class, defining any additional characteristics that are desired, and overriding any inherited characteristics that are not desired.

Figure 6-1. Object Hierarchy



SOMObject

Under SOM's run-time environment, all object classes descend from (are subclasses of) the root object class *SOMObject*, which defines a set of methods and the basic characteristics and behavior common to all objects in the system. These SOM objects are *language neutral*; that is, they can be written in one programming language and used by other objects or applications that were written in another programming language. In theory, an object-oriented application can be developed in several languages, with each object class written in a different language.

Two additional classes, *SOMClass* and *SOMClassMgr*, are provided by SOM. These three classes make up the SOM run-time environment and are found in the SOM.DLL included with OS/2 2.1.

SOMClass

SOMClass is a subclass of SOMObject, which defines the essential behavior common to all SOM object classes. SOMClass is also the metaclass of the SOMObject class. A *metaclass* is a class of classes; it defines the class methods available for a given object class. An object class is an instance of its metaclass. Because SOMClass contains class data and methods that pertain to all SOM object classes, it is, by definition, its own metaclass.

To illustrate the concept of objects, object classes, and metaclasses, consider the example of an automobile factory. The factory plans (metaclass) determine the type of factory, that is, what kind of cars can be built. The factory (object class) then creates the cars (objects).

NOTE

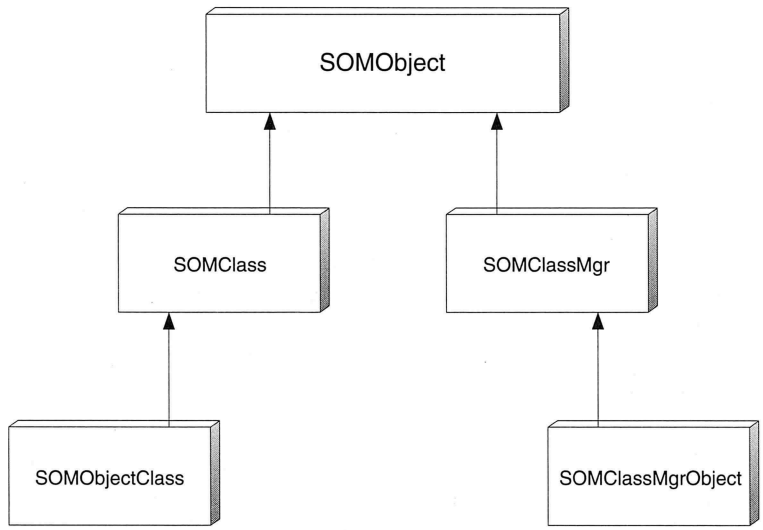
The terms *class*, *object class*, and *class object* are often used interchangeably even though they have different contexts; that is, classes are defined at compilation time while object classes (class objects) are SOM run-time implementations of SOM classes.

SOMClassMgr and SOMClassMgrObject

During SOM initialization, four objects are created (see Figure 6-2 on page 102). The first three are SOMObject, the subclass SOMClass, and the subclass SOMClassMgr. The fourth object is an instance of SOMClassMgr called *SOMClassMgrObject.*

When referenced, SOMClassMgrObject dynamically loads and unloads class libraries, ensures that there is only one (unique) object class for each SOM class, and tracks object instances.

Figure 6-2. SOM Hierarchical Diagram



SOM Programming Features

Three important object-oriented programming features of SOM are inheritance, encapsulation, and polymorphism.

Inheritance

Inheritance, as previously mentioned, is the derivation of new subclasses or *child* classes from existing parent classes. Because child classes inherit the characteristics of their parent classes, any methods defined for a parent class are automatically defined for the child class. Child classes define new behavior by adding new methods to those that they have inherited.

Encapsulation

An object can be viewed in two different ways: through an external (public) view or an internal (private) view. The *public* view determines how other objects or applications can interact with it. The *private* view determines how data and methods are implemented; the actual object implementation is hidden (*encapsulated*) from the public view.

TIP

Although all of a SOM object's definition can be externalized (made public), we recommend that the decision of what to externalize be carefully considered. The reason for this is that the methods then become a permanent part of an object's interface and might make compatibility with future releases more difficult.

Changes to a SOM object's internal implementation does not affect compatibility. One of the advantages of this is that an application that uses SOM objects does not have to recompile every time an object definition changes. This means that full-forward, binary compatibility can be retained when:

- New classes are inserted in the inheritance hierarchy above the current object class.
- Unpublished instance variables are added, changed, or deleted (providing that old methods are still supported).
- Methods are moved up in the class hierarchy.
- A new method is added to the end of the release order. (Note that the release order must remain the same from release to release.)

Polymorphism

The term *polymorphism* means that the same method can be supported or received by several object classes but implemented differently by each one. In SOM, polymorphism is known as a *method override*. This means that a method can have the same parameters and semantics, even though it is implemented by multiple classes.

Polymorphism by inheritance means that methods inherited from a parent class can be overridden by the child class that inherits them.

SOM Run-Time Environment

The SOM run-time environment can be created automatically or by any process that uses it. It consists of three classes that contain the data structures and functions used to define, create, and manage class objects (see "SOMObject" on page 100).

Note that SOM always performs a *per-process* initialization. This means that object classes and their instances cannot be shared across process boundaries.

Defining Object Classes

Because it is important to understand objects and their relationships based upon the needs of the users of your application, the following questions should be answered before defining the object classes:

1. Who are the users?
2. What objects are required by the users?
3. How are these objects related?
4. What tasks will the users perform with these objects?
5. What properties and behaviors should the objects have?

Figure 6-3 provides a simple illustration of the importance of these questions and answers.

Figure 6-3. Illustration Example

Processing Insurance Claims
An insurance company has files that contain information on each customer. These files can be thought of as object folders with one object representing each customer. An accident has been reported and a claim filed. The insurance agent, claims adjustor, and billing clerk (users) require access to the customer's file to process the claim. The Claims Processing application of this company (another object folder) contains several documents: accident forms, appraisals, repair bills, etc. These documents can be considered the data file objects for the customer, which can be adjusted, copied, printed, or discarded (properties and behaviors of the objects). Relationships exist between the customer and the application (both object folders), the customer and the documents, the application and the documents, and between the documents themselves. Some of the tasks performed by the users are as follows: the claims adjuster analyzes the appraisals to determine the total damage; the billings clerk calculates the percentage of reimbursement based on the repair bills and issues a check; the insurance agent uses the customer's address and phone number to report that the claim has been processed and the proverbial check is in the mail.

An actual software model designed with an object-oriented user interface would be much more complicated than this simple scenario. However, a thoroughly prepared design should be a part of any process, large or small.

It is important that the user feel comfortable when interacting with various objects. Therefore, object response should seem natural to the user and similar objects should respond in similar ways. Objects represented visually to the user should be functional and pleasing to the eye. In addition, a choice of interaction mechanisms should be available to users to comply with their skill level, preferred style of interaction (for example, context menus or accelerator keys), and task needs.

Creating a SOM Object Class

Two steps are necessary to create a SOM object class: the class and its relationship to other classes must be defined in a class definition file, and that file must be processed by the SOM compiler.

A *class definition* file is an ASCII file that has an extension of CSC. The class definition is written in a formal specification language called the *SOM Object Interface Definition Language (OIDL)*, which is language neutral and appropriate for any programming language.

Once the class definition file is written, it is processed by the SOM compiler to generate a set of use-specific *binding files* for the object class. These binding files, which in OS/2 2.1 are the SOM C-language bindings, have the same file name (but different extensions) as the class definition files. For example, if a class definition file named SAMPLE.CSC was processed by the SOM compiler, the C-language bindings generated might be named SAMPLE.H, SAMPLE.SC, etc.

The SOM C-language bindings files are used by *client applications*, which create object instances or subclasses, to build class libraries. To create and manipulate a SOM object, a client application must access the public methods of the object by including the public class header (.H) files for that object's class. See "Generating Files with the SOM Compiler" on page 109 and "Using Pointers in Method Invocation" on page 114.

Creating a Class Definition File

All the information necessary to create a new class object is found in the class definition file, which contains the following sections:

- Include section
- Class section
- Parent class section

- Release order section
- Metaclass section
- Passthru section
- Data section
- Methods section

Note that SOM class names are case sensitive wherever used.

Include Section

This section contains a statement to include the class definition file for the parent class of the new SOM object class.

Class Section

This section contains basic information on the new class such as its name, release level, position in the class hierarchy, etc. (see Table 6-1). All new classes are required to have this section in their class definition file.

Table 6-1. Class Section Information

Item	Description
class	All new classes must have a unique name; do not use *SOMxxxx* or *WPxxxx*.
class prefix	Specifies the prefix to be used by the SOM compiler for naming class methods.
external prefix	Specifies the prefix used on all function names. For example, if a function is named *GetInfo* and the external prefix is *extp*, the function name generated by the SOM compiler would be *extp_GetInfo*. When editing source code, it is only necessary to type the function preceded by an underscore character (*__GetInfo*).
file stem	Specifies the file name used by the SOM compiler to generate binding files.
local	Optional. Indicates whether the binding files are to be linked locally. Can be used with or without the *global* option.
major version	Specifies the major version level for the bindings files.
minor version	Specifies the minor version level for the bindings files.

Parent Class Section

This section indicates the parent class of the new class and is a required section.

Release Order Section

This section specifies the order in which the public data and methods of the new object class will be released. It is not necessary to include the release order of methods that

are inherited by the new class. Only new methods or inherited methods that require modification should be defined in this section (see "Encapsulation" on page 102).

TIME SAVER

Complete the Release Order section to save future compilation time. If the new object class is later subclassed, the release order specifications in this section will allow changes to be added to this class without having to recompile the subclass with every change to this class.

Metaclass Section

This section is optional. If a metaclass is specified in this section, it replaces the default metaclass, which is the metaclass of the parent class of this new object class. The new or *implied* metaclass can provide new class methods for the new object class. See "Methods Section" below.

Passthru Section

This section enables blocks of C-language source code to be defined and passed through to the binding files. Generally, this code is used in *typedef* and *#define* statements.

Data Section

This section specifies the instance variables used by the instances of the new object class.

Methods Section

This section lists all methods defined by the new object class, including new and overridden inherited methods.

TIP

To simplify coding, we recommend you separate these definitions into two categories called *new methods* and *overridden methods*.

A typical class definition file structure is illustrated in Figure 6-4 on page 108. Note that comment lines preceded by a "#" are ignored by the SOM compiler and are not

displayed in the binding files. Comment lines preceded by a "--" are passed as a comment block to the interface definition (.SC) file.

Figure 6-4. Structure of a Class Definition File

```
#
# Include the class definition file of the parent class
#
include sc
#
# Define the new class
#
class: claim,
       file stem = claim,
       external prefix = claim,
       class prefix = claim,
       major version = 1,
       minor version = 1,
       local;
-- Claim is an Automotive Policy Claim (type of class)
-- It is derived as follows:
--                          SOMObject
--                           - Company
--                             - State
--                               - Policy type
#
# Indicate the parent class
#
parent: SOMObject;
#
# Specify the release order
#
release order: NewClaim;
#
# Specify the passthrus using this identifier, passthru:language
#
passthru: C.ph;                          /* PRIVATE definitions */

   typedef struct _ABSDATA
   {
     ULONG cbObjData;
     CHAR szClass[1];
   } ABSDATA;
   typedef ABSDATA*PABSDATA;

   #define SIZEOF_ABSDATA(x)(sizeof(ABSDATA)+ strlen(x-szClass))
   typedef struct _ABSFIND
   {
     PULONG pHandle;               /* Array of handles for this folder      */
     ULONG cHandles;               /* Count of remaining handles for folder */
     PABSDATA pAbs;                /* Pointer to work area memory           */
   } ABSFIND;
   typedef ABSFIND *PABSFIND;

#define ABS_ID_TITLE (ULONG)1
#define ABS_ID_STYLE (ULONG)2
endpassthru:

#
# Define the instance data for the new class
#
```

Structure of a Class Definition File (continued)

```
data: ClaimNo;
-- This instance data is insured
#
# Define the new methods to be used with the new class
#
methods:
VOID Claim(), Class;
-- Method:       VOID WrClaim
-- Method type: Private (private or public)
-- Purpose:      Track claim
#
# Specify all overridden inherited methods
#
override ClaimHandle;
-- Override: Deduct for high mileage
-- Purpose:  Tie in damage estimate to worth of car
```

Generating Files with the SOM Compiler

The SOM compiler uses the class definition file as input to generate C-language binding files, which are then used to define the new object class to the Workplace Shell or to other object classes. Note that the other classes can become subclasses of the original class; that is, they can inherit the characteristics of the original class, which would then become the parent of these other classes.

All binding files have the same file name as the class definition file. The binding file and class implementer file extensions are shown in Table 6-2.

Table 6-2. Binding and Class Implemeter Extensions

Extension	Binding File Description
H	Public header file; used by applications that use the object class.
PH	Private header file; provides usage bindings for private methods used by object class.
SC	Class definition file that is language independent.
PSC	Private core file (language independent); contains private interface parts.
Extension	**Class Implementer File Description**
C	Template C file. Code can be added to this file to implement the object class.
DEF	OS/2 DLL module definition file. Contains exports needed to implement object class.
IH	Implementation header file; provides macros, functions, etc. for class implementation.

Binding files are input to a C-language compiler to generate object code. The object code is then linked to create a dynamic link library (DLL), which implements the object class.

Building SOM Class Libraries

The file types *.SC* and *.H* are required to build SOM class libraries. The .SC files establish the relationships between object classes and are used by the SOM compiler to process class definition files. The .H files resolve references to methods defined in other files and are used by the C-language compiler to process and build the class libraries.

To build a class library, perform the following steps:

1. Set up the class definition (.CSC) files for all parent and ancestor classes (and their related classes).
2. Compile these .CSC files using the SOM compiler.
3. Set up the .CSC files for the new object class and its metaclass.
4. Compile the metaclass .CSC file using the SOM compiler.
5. Compile the .CSC file of the new class using the SOM compiler.
6. Compile the .C template (implementation) files in any order using the C-language compiler.

C File Template Stub Procedures

The C-language program template generated by the SOM compiler contains stub procedures for all new and override methods specified in the class definition file. A *stub procedure* is a routine that contains only an entrance and exit to a function; it does not perform any functions. Your code for stub-method procedures is added to the template to enable class implementation, for example, *void CclaimGet(int x);*.

The implementation header (.IH) file, which is automatically included in the program template, contains the data structure, macros, and functions used to access methods and data for object instances. The .IH file provides stubs for all methods defined in the Methods section of the class definition file. An example of a method stub is shown in Figure 6-5.

Figure 6-5. Method Stub

```
SOM_Scope void SOMLINK <methodname>(<classname> *somSelf)
 {
  <classname>Data *somThis = <classname>GetData(somSelf);
  <classname>MethodDebug ("<classname>","<methodname>");
 }
```

Note that if you alter a method definition, the SOM compiler will not merge the change back into your C source code; you must do it manually.

SOM Methods and Macros

Methods for SOM objects use the prefix *som*. Contents, data types, and pointers to functions (including macro names) use the prefix *SOM*. Object methods are invoked by preceding the method name with an underscore (_) macro. Underscored data-name, method-name, and class-name macros are defined in the .H public class header file.

SOM macros are used to instantiate objects, access object variables, and invoke object methods. For example, the macros, *SOM_Scope* and *SOMLINK*, are used by the C-language binding files to invoke methods to implement classes.

There are two type of SOM macros: class specific and general.

Class-Specific Macros

Class-specific macros, which are automatically generated by the SOM compiler, provide prototypes for class methods, class function, and class instance data. These macros can be included in the implementation header (.IH) files. Because the header (.H) file is included in the .IH file, any macros in the .H file can be used by .IH file class implementers.

The class-specific macros are described in Table 6-3 on page 112.

General SOM Macros and Functions

General SOM macros (which are defined in the header file *SOM.H*) and their invoked functions support ID manipulation, object information retrieval, debugging, and error handling. These macros are available to class implementers such as class methods and client applications.

Table 6-3. Class-Specific Macros

Macro	Definition
<classname>GetData	Is defined only in the .IH file. Retrieves data for the object instance.
<classname>MethodDebug	Is automatically generated and placed in the .IH file. Traces a method when supplied with the class name and method name. Note that if *SOM_TraceLevel* is set to *1* or *2*, a message is produced each time a method is entered. (See **TIP**.)
<classname>New	Instantiates objects. The function *<classname>NewClass* is defined in the .H file and invoked automatically when a new object class is created by this macro.
<classname>Renew	Instantiates data.
get_<instance variable>	Retrieves instance data for the object instance (somSelf) of an object class.
parent_<methodname>	Invokes parent methods. This macro is defined only in the .IH file and used for object class implementation; it cannot be used in client code.
SOM_ParentResolve	Invokes parent methods. This macro is defined only in the .IH file and used for object class implementation; it cannot be used in client code.
SOM_Resolve	Invokes methods.
SOM_ResolveNoCheck	Invokes methods without verifying that the object class matches the class method being called.

TIP

When writing retail code, suppress the method tracing output by creating a #define statement after the #include statement for *<classname>.IH* in your .C file. This will save a lot of disk space. Use the format:

```
#define <classname>MethodDebug(c,m) SOM_NoTrace(c,m)
```

Manipulating SOM IDs

A number that uniquely represents a string is called an *ID*. SOM IDs are used to identify class names, method names, and descriptors. To manipulate these IDs, SOM provides the set of macros and functions shown in Table 6-4. When any of these macros are first invoked, the SOM ID is automatically converted into an internal ID representation. Because of this, SOM IDs are of a unique data type called *somid*.

Table 6-4. SOM ID Manipulators

Macros	Functions
SOM_CheckID	somBeginPersistentIds
SOM_CompareIDs	somEndPersistentIds
SOM_IDFromString	somRegisterId
SOM_StringFromID	somSetExpectedIds
	somTotalRegIds
	somUniqueKey

Retrieving Object Information

To retrieve object information, invoke the macro or method that created the object. For example, invoking the macro *SOM_GetClass* returns a pointer to the class object. The method *somGetInstanceSize* returns the total size of the object. The method *somGetInstancePartSize* returns the size of the instance variables contributed by a particular class object.

Replacing SOM Functions

The SOM run-time environment uses replaceable SOM functions that perform character output, DLL management, memory management, and error handling. By providing your own default SOM functions, the functions in Table 6-5 can be replaced (overridden).

Table 6-5. Replaceable SOM Functions

Character Output	Error Handling
SOMOutCharRoutine	SOMError
DLL Management	**Memory Management**
SOMLoadModule	SOMFree
SOMDeleteModule	SOMCalloc
SOMClassInitFuncName	SOMMalloc
	SOMRealloc

Debugging in SOM

Four macros can be invoked by the SOM run-time library to assist in debugging an application. The debugging output can be generated or suppressed, depending upon the setting of the global control variables within each macro (see Table 6-6).

Table 6-6. Debugging Macros

Debugging Macro	Global Variable
SOM_Assert	SOM_AssertLevel
SOM_Expect	SOM_WarnLevel
SOM_TestC	SOM_WarnLevel
SOM_WarnMsg	SOM_WarnLevel

To generate character output, SOM provides the function *somPrintf*, which can be used in place of *printf*. Because SOM functions are replaceable and more flexible than C-library functions, we recommend that your SOM class implementations use SOM functions whenever possible.

SOM errors are classified by the severity levels, *SOM_Ignore*, *SOM_Warn*, and *SOM_Fatal*. Error levels are indicated by the low-order digit in the SOM error code. SOM errors are handled by the SOM run-time library. This library invokes the macros, *SOM_ERROR* and *SOM_TEST*, which call a replaceable SOM procedure called *SomError*. SomError produces an error message and can, if appropriate, end the process where the error occurred.

Invoking SOM Methods

To invoke a SOM method, precede the method name with an underscore (_) character and include an object pointer as the first parameter (see "Using Pointers in Method Invocation" below). To invoke an instance method, make sure that the pointer is set to an instance object. To invoke a class method, ensure that the pointer is set to a class object.

To access data within an object instance, precede the name of the data element with an underscore (_) character.

Method Invocation Pointers

The pointer *somSelf* is always the first parameter in a method invocation. Because methods can be implemented differently on different objects (polymorphism), a pointer to the object instance (the object to be operated on) is required. To set the pointer to an instance of an object class, use the format: *<classname> *object:*.

The pointer to data for an object instance is called *somThis*. The SOM compiler automatically generates the class data type *<classname>Data* and places it in the .IH file.

Classifying an Object

As explained in Chapter 5, there are three basic object classes (data, device, and container), each of which is distinguished by a particular purpose. However, any object class can contain other object classes. Your application should define its objects and their properties in terms of these basic object classes.

Selecting the Appropriate Object Views

The object views selected for display should be the ones most appropriate for the needs and preferences of the users. In Figure 6-3 on page 104, for example, the claims adjustor would need to create, view, and modify repairs estimates and work orders while the billing clerk might need only to view the accounting records and print checks.

Determining the Appropriate Action Choices

It is important to determine what action choices are needed by the user for each object type. Some objects are relatively fixed in nature, thereby limiting the action choices. Other objects are more open-ended and can have a multitude of potential action choices. For example, as described in Figure 6-3, the claims adjuster will most likely need to add, change, or delete information from the appraisal. In addition, some math calculations will be necessary to determine the total damage and the percent of reimbursement.

Chapter 7.
Using the Workplace Shell Environment

The Workplace Shell (WPS) is built upon OS/2's *Presentation Manager*, which is a set of system services that provide screen management, graphics-based presentation functions, and communication facilities. WPS provides an environment where all services are task oriented; therefore, users do not have to understand the complex processes and concepts that make up OS/2 2.1, such as configuring the system, managing files, and starting applications. Because they are not involved with the functioning of the operating system, their learning time is lessened and tasks can be performed easily and quickly.

The file system in WPS is transparent to users so it is not necessary for them to understand the workings of disk drives or directories. However, they should know that objects such as applications and files can be put into a folder, which can be labeled with any name that seems appropriate. Folders can store and display objects or other folders. This system is less demanding on users and allows them to arrange objects (regardless of where the objects physically reside) to suit their own needs.

For consistency, all objects in WPS are designed to be data oriented and to be acted upon by users in a similar manner (for example, a file can be manipulated in the same way as an application). Also, there is no difference in how the same application is used, whether it is on a hard disk, CD-ROM, or network server.

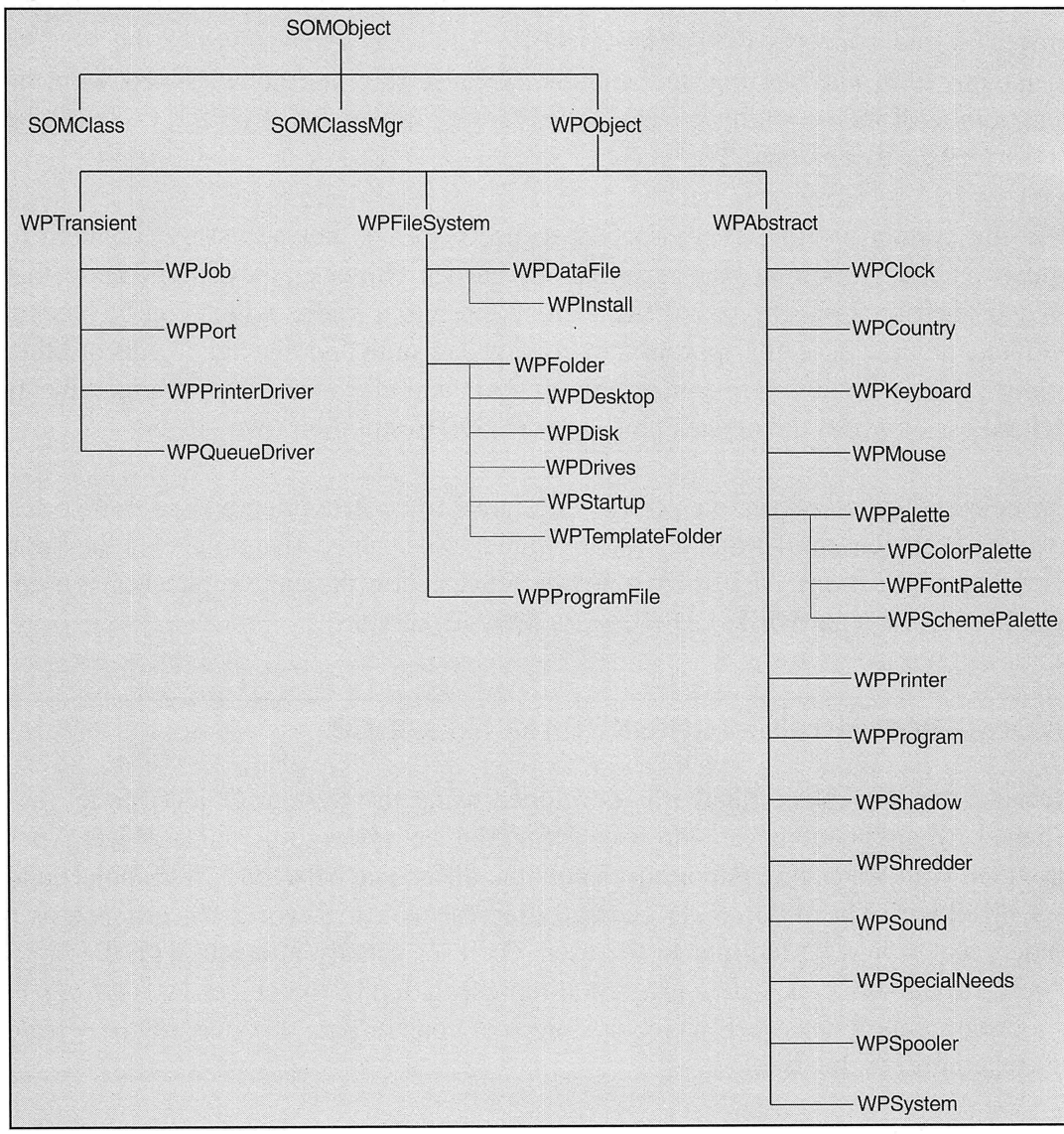
Workplace Shell Programming Interface

Because the Workplace Shell was developed using the System Object Model (see Chapter 6), applications can add user-defined object types to the class object types provided with WPS. It is important to note the difference between a WPS object and a SOM object. The *WPS object* generally represents a real-world object (such as a folder) that would be familiar to the user. The user usually interacts with the WPS object through an icon, which is a visual representation of the object. A *SOM object* is a programming construct, which is a piece of code or data that may not be visible or meaningful to the user.

Workplace Shell Classes

Every WPS object has a corresponding WPS class. WPS classes are SOM classes. As described in Chapter 6, all SOM classes are derived from the root SOM class, SOMObject. In the same way, all WPS classes are derived from a root WPS class called *WPObject*, which is also derived from the root SOM class, SOMObject. An example of the WPS inheritance hierarchy is shown in Figure 7-1.

Figure 7-1. WPS Inheritance Hierarchy



Rules that apply to SOM classes also apply to WPS classes with one exception: WPS methods cannot be called by applications. Most WPS objects are created (instantiated) as unique entities with specific attributes by one of the following:

- The OS/2 installation process
- A batch file
- The WPS Class List Object

WPObject, WPAbstract, WPFileSystem, and *WPTransient* cannot be instantiated because they define common characteristics and behavior for subclasses. Like their SOM counterparts, WPS classes are defined with the Object Interface Definition Language (OIDL) and WPS class libraries are built by the SOM compiler.

WPS Storage Classes

All WPS classes are derived from three WPS storage classes, which are derived from WPObject. A *storage class* defines the methods used to store and retrieve control information and instance data for subclasses. The following storage classes are provided with the Workplace Shell:

WPAbstract	Stores programs, drives, etc. in the OS2.INI file.
WPFileSystem	Stores file and folder objects as files and directories on the disk.
WPTransient	Stores transient objects only during the execution of a program.

The storage classes, WPAbstract and WPFileSystem, are considered *persistent* objects. WPS objects are also persistent; their existence is controlled by the base (storage) classes. The state and instance data of a persistent object are constant (unchanging). A *nonpersistent* object is dependent upon some relationship to the user; its state and data are temporary.

Theoretically, a new storage class can be created by subclassing the root class (WPObject). However, we recommend that you create new WPS classes from the existing storage classes or their descendants. Select the class that is closest to the requirements you have for your subclass as the base from which to begin.

Implied WPS Metaclasses

All WPS objects have *implied* metaclasses, that is, their metaclasses are defined in their class definition files. Because additional class definition files are not required for these metaclasses, the number of files needed to build an object is reduced. The WPS

metaclasses define the WPS class methods, which are prefixed by *wpcls*. These methods define the WPS object class properties (for example, the default instance attributes of an object).

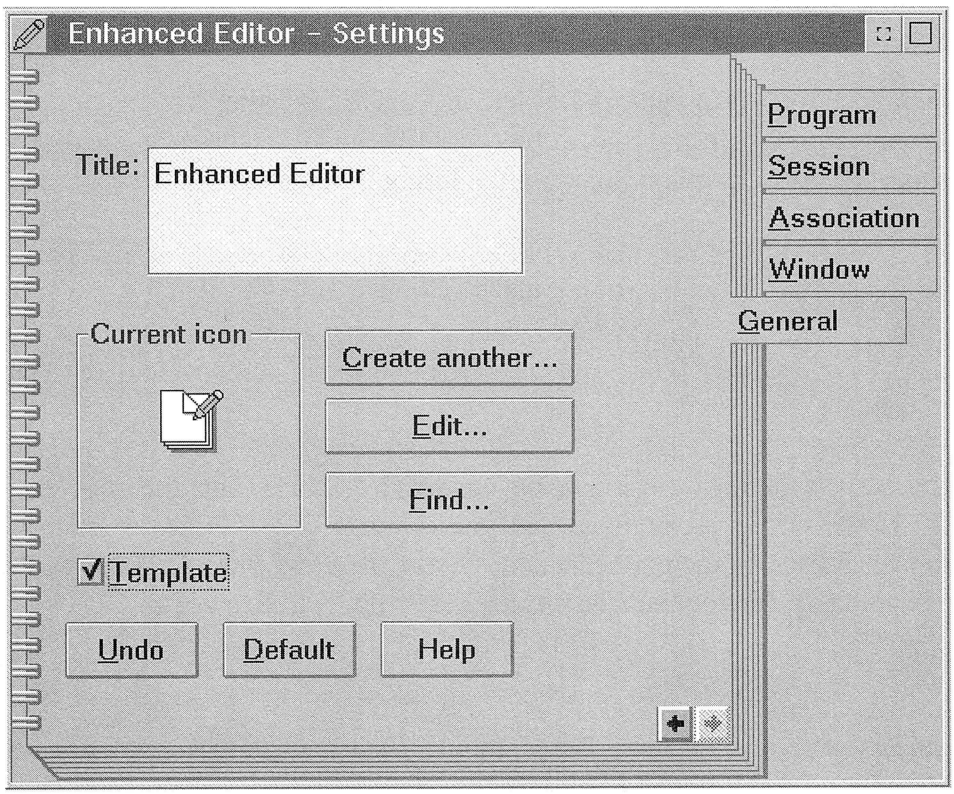
Defining WPS Object Classes

WPS object classes are defined in the same way as SOM object classes. The object relationships must be determined, the appropriate actions identified, and the corresponding methods defined in the class definition file.

Modifying Pages in a Settings Notebook

All WPS objects have a *Settings notebook*, which contains a tab named **General**. When selected, this tab displays a General page, which describes the general properties associated with the object (see Figure 7-2).

Figure 7-2. Settings Notebook



Some Settings Notebook methods allow new pages to be added to, or removed from, a class. For example, WPS builds the Settings notebook of an object by calling *wpAddSettingsPages*.

To add new pages to a Settings notebook that has been inherited from an object's ancestors, you must override the existing wpAddSettingsPages by replacing it with a new wpAddSettingsPages method that calls *wpInsertSettingsPage* (see Figure 7-3). The wpInsertSettingsPage method will then insert the new pages into the object's notebook.

Figure 7-3. Adding a Page to a Settings Notebook

```
SOM_Scope ULONG SOMLINK MyObject_wpAddAnotherPage(MyObject *somSelf,
                                                  HWND hwndNotebook)
PAGEINFO pageinfo

/************************ Set up PAGEINFO structure ***************************/

pageinfo.ulcb                 = sizeof(PAGEINFO);  /* Size of PAGEINFO          */
pageinfo.hwndPage             = hwndPage;          /* Handle of Page            */
pageinfo.ppfnwp               = CommentDlgProc;    /* Window procedure          */
pageinfo.ulresid              = vhmodMRI;          /* Resource identity         */
pageinfo.pCreateParams        = somSelf;           /* Pointer to CreateParams   */
pageinfo.usdlgid              = IDD_FILE;          /* Dialog identity           */
pageinfo.usPageStyleFlags     = 0;                 /* Notebook control          */
pageinfo.usPageInsertFlags    = BKA_FIRST;         /* Notebook control          */
pageinfo.usReserved1          = 0;                 /* Reserved; must be 0       */
pageinfo.pszName              = vszrEmpty;         /* Pointer to pg # string    */
pageinfo.usidDefaultHelpPanel = DATAFFILE3RES;     /* Default Help panel id     */
pageinfo.usReserved2          = 0;                 /* Reserved; must be 0       */
pageinfo.pszHelpLibraryName   = vsrHelpLib         /* Pointer to Help filename  */
pageinfo.pHelpSubtable        = HelpSubTbl         /* Pointer to Help subtable  */
pageinfo.hmodHelpSubtable     = HelpModule         /* Help subtable mod handle  */
pageinfo.usPageInsertID       = ID_NoteBook        /* Page identity             */

return = _wpInsertSettingsPage(somSelf, hwndNotebook, &pageinfo);
```

To add a new page to the top of a Settings notebook, call *parent_wpAddSettingsPage* before calling *wpAddAnotherPage*. To add a new page to the bottom of the notebook, call wpAddAnotherPage before calling parent_wpAddSettingsPage (see Figure 7-4).

Figure 7-4. Positioning a Notebook Page

```
HWND SOMLINK MyObject_wpAddSettingsPage(MyObject *somSelf, HWND hwndNotebook)
}
if (parent_wpAddSettingsPages(somSelf, hwndNotebook)&& _wpAddAnotherPage
   (somSelf, hwndNotebook))
{
   return(TRUE);
}
else
{
return(FALSE);
}
```

To remove an inherited page, you must override the method that inserted it and return SETTINGS_PAGE_REMOVED. To remove or replace the General page, override the methods wpAddSettingsPages and *wpAddObjectGeneralPage*. The override to the wpAddObjectGeneralPage method returns SETTINGS_PAGE_REMOVED (to remove the General page) or calls wpInsertSettingsPage (to replace the General page).

For more information on the Notebook control, refer to the *SAA: Common User Access Advanced Interface Design Reference* in the OS/2 2.0 Technical Library.

Using Object Usage Methods

The Object Usage methods *wpFindUseItem*, *wpAddToObjUseList*, and *wpDeleteFromObjUseList* enable an object to track its resources and the manner in which they are used.

All WPS objects use an *in-use* list, which provides information such as memory allocation or the number of open views. This in-use list is actually a linked list of USEITEM data structures, which consist of an item type and a pointer.

The following item types can be added to an in-use list:

- USAGE_OPENVIEW
- USAGE_MEMORY
- USAGE_RECORD

Their corresponding data structures, which are defined in the .H files provided with the OS/2 2.x Toolkit, are as follows:

- VIEWITEM
- MEMORYITEM
- RECORDITEM

To open a view of an object, call *wpOpen*. This method calls *wpAddtoObjUseList*, which adds a USAGE_OPENVIEW item to the in-use list of the object.

To close an object's view, call *wpClose*, which calls *wpDeleteFromObjUseList* to remove a USAGE_VIEW item from the object's in-use list.

To allocate memory for an object, call *wpAllocMem* (see Figure 7-5). This method calls wpAddtoObjUseList to add a USAGE_MEMORY item to the object's in-use list.

To free the object's memory, call *wpFreeMem*. This method calls wpDeleteFromObj-UseList to remove a USAGE_MEMORY item from the in-use list.

Figure 7-5. Allocating Memory for an Object

```
pItem = MyObject_wpAllocMem(MyObject,sizeof(MemRequired),NULL);

if (!pItem)
{
    return(FALSE)
)
BOOL _wpClose(MyObject *somSelf);
```

If you forget to free memory, OS/2 2.1 will do it for you; however, it is best not to rely on this as a cleanup method.

To insert an object into a CUA container control window, call *wpCnrInsertObject*, which adds a USAGE_RECORD item to the object's in-use list. To remove an object from a container window, call *wpCnrRemoveObject*. This method calls wpDelete-FroObjUseList to remove a USAGE_RECORD item from the in-use list.

To determine how an object is currently used, call *wpFindUseItem*. This method searches the object's in-use list and returns a pointer to the USEITEM structure that matches the specified criteria (item type).

Using the Drag and Drop Feature

This feature enables files to be manipulated easily in OS/2 2.1; simply use the mouse to drag them to the desired action object. For example, to print a file, drag it to the printer object; to delete a file, drag it to the shredder object. This eliminates the need to memorize long and complicated lists of commands that must be typed in a precise sequence to achieve a particular action.

Methods that support the "drag and drop" feature are called *direct manipulation* methods. The target object is notified by the Workplace Shell when an object is dropped on it. WPS then calls the target object's Direct Manipulation methods to process the source object that has been dragged and dropped. (Note that you usually do not need to incorporate "drag and drop" into your code because it is handled by the Workplace Shell.)

Although target objects can process more than one type of object, they cannot process every object type. For example, a shredder object can destroy many different file types but cannot destroy a startup file or other important system file. Because of this, target objects must be able to determine which source objects they can handle and should notify the user when a source object cannot be handled.

Using Setup Methods

The following Setup methods support object initialization and destruction:

- wpSetup
- wpScanSetupString
- wpInitData
- wpUnInitData
- wpFree
- wpRestoreData
- wpSaveData

An object class defines the setup or *KEYNAME* variables and values that control the behavior of an object.

Although KEYNAMEs have default values, new values can be specified in a setup string by using the format that follows.

```
pszSetupStrings="TITLE=objectname;
ICONFILE=objectname.ICO;
OBJECTID=<objectid>";
```

Note that the caret (^) character is used as an escape character. For literal commas, use "^,". For literal semicolons, use "^;", as shown:

```
SET DOS_VERSION = 4^, 0^, 0; TITLE = MyDOSPgm
```

To specify KEYNAME values in a setup string when an object is created, call *WinCreateObject*. To specify KEYNAME values when the behavior of an existing object is changed, call *WinSetObjectData*.

To process the KEYNAMEs, call *wpSetup* (see "Using Object Instance Functions" on page 133). KEYNAMEs and values supported by WPObject are listed in Table 7-1.

Table 7-1. Supported KEYNAMEs and Values

KEYNAME	Value	Description
CCVIEW	yes or no	If *yes*, a new object view is created each time the **Open** option is selected. If *no*, the open view of the object is displayed when **Open** is selected.
HELPPANEL	id	Sets the default help panel of the object, which is equivalent to calling *wpSetDefaultHelp*.
ICONFILE	filename	Sets the title of the object (equivalent to calling *wpSetTitle).*
ICONPOS	x, y	Sets the object's initial icon position in a folder in *x* and *y* percentage coordinates.
ICONRESOURCE	id, module	Sets the icon of the object (equivalent to calling *wpSetIconData).* Use the icon resource ID in the DLL located in the object's class definition file for the *id* value.
MINWIN	desktop, hide, viewer	When the **Minimize** button is selected, if this value is set to *desktop*, the view is minimized to the Desktop; if set to *hide*, the view is hidden; if set to *viewer,* the view is minimized and can be viewed in the *Minimized Window Viewer.*
NOCOPY	yes or no	If *yes*, user cannot copy object (equivalent to calling wpSetStyle with *OBJSTYLE_NOCOPY).* If *no*, user can copy object.
NODELETE	yes or no	If *yes*, user cannot delete object (equivalent to calling wpSetStyle with *OBJSTYLE_NODELETE).* If *no*, user can delete the object.
NODRAG	yes or no	If *yes*, user cannot drag object (equivalent to calling wpSetStyle with *OBJSTYLE_NODRAG).* If *no*, user can drag the object.
NOMOVE	yes or no	If *yes*, user cannot move object (equivalent to calling wpSetStyle with *OBJSTYLE_NOMOVE).* If *no*, user can move the object.
NOPRINT	yes or no	If *yes*, user cannot print the object (equivalent to calling wpSetStyle with *OBJSTYLE_NOPRINT).* If *no,* user can print the object.
NORENAME	yes or no	If *yes*, user cannot rename the object (equivalent to calling wpSetStyle with *OBJSTYLE_NORENAME).* If *no*, user can rename the object.
NOSHADOW	yes or no	If *yes*, user cannot create a shadow (equivalent to calling wpSetStyle with *OBJSTYLE_NOSHADOW).* If *no*, user can create a shadow of the object.
NOTVISIBLE	yes or no	If *yes*, the object is not visible to the user (equivalent to calling wpSetStyle with *OBJSTYLE_NOTVISIBLE).* If *no*, the object is visible.

Supported KEYNAMEs and Values (continued)

KEYNAME	Value	Description
OBJECTID	<name>	Sets a persistent string identifier ID for the object, which is shown as a unique string preceded by a "<<" and followed by a ">>". **Note:** Do not use *WPxxxx*!
OPEN	default or settings	If the value is *default*, the default view is opened when the object is created or when *WinSetObjectData* is called. If *settings*, the settings view is opened when the object is created or when WinSetObjectData is called.
TEMPLATE	yes or no	If *yes*, user can create a template (equivalent to calling wpSetStyle with *OBJSTYLE_NOTEMPLATE*). If *no*, user cannot create a template of the object.
TITLE	<title>	Sets a title of an object (equivalent of calling *wpSetTitle*).
VIEWBUTTON	hide, viewer	If set to *hide*, the object view displays a **Hide** button instead of a **Minimize** button. If set to *viewer*, the object view displays a **Minimize** button instead of a **Hide** button.

To effect a change on an icon for an existing object, call *WinQueryObject*, which uses the object's OBJECTID to receive a handle to the object; then call WinSetObjectData with the *ICONDATA* KEYNAME value specified in the setup string.

Using Save/Restore State Methods

The following Save/Restore State methods affect the class definitions of persistent objects:

- wpSaveDeferred
- wpSaveImmediate
- wpSave/wpRestoreData
- wpSave/wpRestoreLong
- wpSave/wpRestoreState
- wpSave/wpRestoreString

WPS calls *wpRestoreState* when an object is activated. When an object is closed, dormant, or the system is shut down, WPS calls *wpSaveImmediate* and *wpSaveState*.

Note that when a critical instance variable is changed, wpSaveImmediate should be called. For better performance, use *wpSaveDeferred* instead of wpSaveImmediate.

To save or restore data to an object:

1. Override wpSaveState, and then call wpSaveData, wpSaveLong, or wpSaveString, depending on the instance data type associated with the object.
2. Override wpRestoreState, and then call wpRestoreData, wpRestoreLong, and wpRestoreString.

Understanding Pop-Up Menus

A *pop-up menu*, which is contained in a pop-up window associated with an object, consists of a set of selectable actions that can be performed on that object. The contents of a pop-up menu depend on the context of the object associated with it. The selectable actions can require additional pop-up or *conditional cascade* menus, if additional action choices are associated with them.

The following Pop-Up Menu methods support selectable actions:

- wpFilterPopupMenu
- wpModifyPopupMenu
- wpInsertPopupMenuItems
- wpQueryStyle

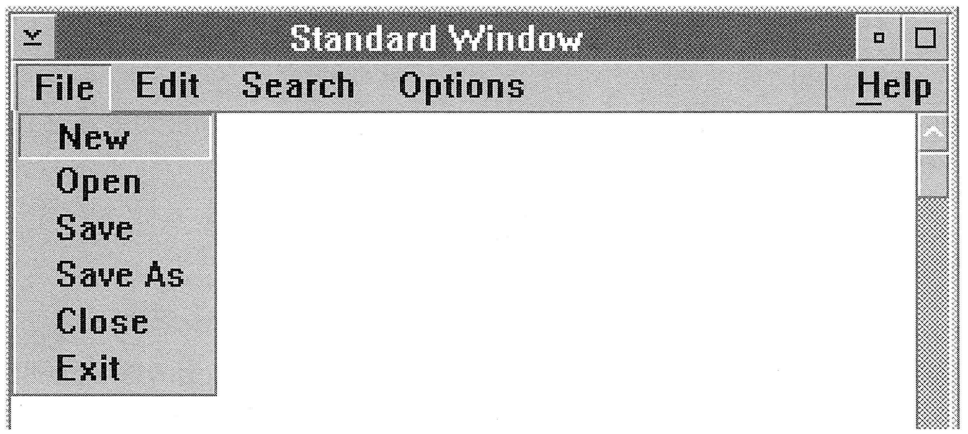
A pop-up menu is built by calling *wpFilterPopupMenu* and *wpModifyPopupMenu*. To add new options to a pop-up menu, override wpModifyPopupMenu and have the new method call *wpInsertPopupMenuItems* (see Figure 7-6).

Figure 7-6. Adding a Pop-Up Menu Item

```
SOM_Scope BOOL32 SOMLINK MyObj_wpModifyPopupMenu(MyObj *somSelf, HWND hwndMenu,
                                                 HWND hwndCnr, ULONG iPosition)
{
   MyObjData *somThis = MyObjGetData(somself);
   if (usFlag & PRINT_MENU)
      wpInsertPopupMenuItems(somSelf, hwndMenu, (ULONG)0, hModule,
                             MyObj_OPENMENU, ID_OPEN);
}
rc = parent_wpModifyPopupMenu(somSelf,hwndMenu,hwndCnr,iPosition);
```

Because pop-up menus are inherited with a standard set of menu items (see Figure 7-7 on page 128), it is often necessary to add or remove items from the menu. To add a menu item, override wpModifyPopupMenu and call wpInsertPopupMenuItems. To remove an item, override wpFilterPopupMenu. This override will *mask* (hide) the flag associated with the menu item.

Figure 7-7. Standard File Pop-Up Menu



To ensure that menu items are not added back into the menu after deletion or deleted after being added, call *parent_wpFilterPopupMenu* before calling an *inherited* wpFilterPopupMenu.

Working with Class-Specific Menu Items

To add a class-specific item to a pop-up menu, follow the override process for adding new menu items, define the new item in a resource file, and assign it an ID. To avoid conflict with other WPS IDs, IDs for class-specific menus and menu items must have a value greater than *WPMENUID_USER*.

To remove a class-specific menu item, override the filtering method that returns the flag associated with that item.

Adding Items to Conditional Cascade Menus

To add an item to a conditional cascade menu, which is a submenu of a pop-up menu, specify the ID for the conditional cascade menu when calling the method wpInsertPopupMenuItems. Note that this method requires a handle to the defined menu resource's module and the ID for the menu resource.

To display the conditional cascade menu, set the style and default selection of the menu. This will also display the small push button associated with the menu.

Defining New Action Choices

When new actions are defined for a pop-up menu, they must be able to function when they are selected by the user. To support the action processes, override the Pop-Up Menu methods, *wpMenuItemSelected* and *wpMenuItemHelpSelected*. The Workplace Shell will call one of the following Pop-Up Menu methods when the corresponding menu item is selected from a pop-up menu:

- wpClose
- wpCopyObject
- wpCreateFromTemplate
- wpCreateShadowObject
- wpDelete
- wpDisplayHelp
- wpHide
- wpMoveObject
- wpOpen
- wpPrintObject
- wpRestore

Object Shadows

An object shadow is a representation of another object, as explained in Chapter 6. To create an object shadow, hold the Ctrl and Shift keys down while dragging the object to the desired location, or select **Create shadow** from the pop-up menu. Either action will call *wpCreateShadowObject*.

Using Open Views

An *open view* is a view of an object (for example, the icon or settings view) that is displayed in a conditional cascade menu when a user selects the mini push button called **Open View**.

The following set of predefined open views are provided by the Workplace Shell:

- OPEN_CONTENTS
- OPEN_DEFAULT
- OPEN_DETAILS
- OPEN_HELP
- OPEN_RUNNING

- OPEN_SETTINGS
- OPEN_TREE
- OPEN_USER

To define a new open view, add the new menu item to the Open View submenu, override wpMenuItemSelected and wpOpen, and call *WinCreateStdWindow* or *WinLoadDlg* to create and open a new window for the view.

TIP

To conserve WPS resources when dealing with large, unreliable code blocks, use *WinStartApp* to start a separate process for an application view. This will move most of the application code out of the WPS process.

After the new window is opened, call wpAddToObjUseList to add a USAGE_VIEW item to the in-use list of the object. Finally, call wpRegisterView to register the new view.

Using a Details View

A *details view* is a container control window for a particular object class that contains information arranged in columns. To build headings for the columns, override the *wpclsQueryDetailsInfo* method.

Each row of data (record) contains specific information about an object instance of that object class. To inherit detail records from ancestor classes, contiguous blocks of memory are needed. To obtain these blocks, which can vary in length, call the object's *wpQueryDetailsData* method. This method will then call its parent's wpQueryDetailsData, which will call *its* parent's wpQueryDetailsData, and so on. A pointer will be moved to the end of the last block added to the record to indicate where the next block should be added.

A linked list of *CLASSFIELDINFO* data structures contains the format for the data of a record. This list is inherited in much the same way as a record, that is, by repeated calls to each ancestor's wpclsQueryDetailsInfo method. The CLASSFIELDINFO data structures are defined in the .H files provided with the OS/2 2.x Toolkit. Note that these structures are *static* and cannot be changed at runtime.

To build a new record for the object, override wpQueryDetailsData. To define a new data format for a record, override wpclsQueryDetailsInfo.

Obtaining Object Information

The following Object Information methods enable the setting and querying of information that is associated with an object:

- wpQueryDetailsData
- wpSet/wpQueryDefaultHelp
- wpSet/wpQueryDefaultView
- wpSet/wpQueryIcon
- wpSet/wpQueryIconData
- wpSet/wpQueryStyle
- wpSet/wpQueryTitle

Defining Object Behavior

A WPS object's behavior is defined by the following object instance styles, which define object behavior:

- OBJSTYLE_NOCOPY
- OBJSTYLE_NODELETE
- OBJSTYLE_NODRAG
- OBJSTYLE_NOMOVE
- OBJSTYLE_NOPRINT
- OBJSTYLE_NORENAME
- OBJSTYLE_NOSHADOW
- OBJSTYLE_NOTDEFAULTICON
- OBJSTYLE_NOTVISIBLE
- OBJSTYLE_TEMPLATE

To change an object style *after* the object has been created, call wpModifyStyle.

Object Templates

The main means of creating object instances is through the use of object templates, which are automatically created in the Templates folder when an object is registered.

A *template* is a pattern of an object that can be reused to create an instance of the object. This is accomplished by dragging and dropping the template, an action that can also be accomplished with the **Create another** option.

The Templates folder contains a template for each object whose *template checkbox* has been activated in the General page of the Settings notebook for the object. To deactivate an object's template state, deselect the object's template checkbox.

Registering a new object by calling *WinRegisterObjectClass* prevents the removal of the new template from the Templates folder. To prevent the automatic creation of a template when you are registering a class, the *wpclsQueryStyle* method should return *CLSSTYLE_NEVERTEMPLATE.*

Using Objects in the Workplace Shell

A WPS object class is activated when the class is registered with WPS and instantiated. The methods that enable this to occur are provided by SOM and called by the Workplace Shell. Therefore, the Workplace Shell can be considered the *client* of the WPS objects in the same way that an application can be the client of the SOM objects. (Note that applications are not clients of the WPS objects and, therefore, cannot call them directly).

The following functions have been provided by WPS to allow an application to change the objects on the Desktop (or the Desktop itself), when necessary:

- WinCreateObject
- WinDeregisterObjectClass
- WinDestroyObject
- WinEnumObjectClasses
- WinFreeIcon
- WinLoadFileIcon
- WinQueryObject
- WinResigterObjectClass
- WinReplaceObjectClass
- WinRestoreWindowPos
- WinSetFileIcon
- WinSetObjectData
- WinShutdownSystem
- WinStoreWindowPos

Object Class Functions

To register an object class with the Workplace Shell, call WinRegisterObject. The class name of the object must match the name specified in the DLL module definition file (see "Generating Files with the SOM Compiler" in Chapter 6). The registered class will be maintained in OS2.INI and cached when the system is initialized.

TIP

To free up cache memory, remove registered classes when they are no longer needed.

The INI.RC script file defines the initial Desktop and illustrates the use of most setup strings. Use this script file (look in the PM-Install Object section) and the MAKEINI.EXE utility to create your own customized Desktop layouts.

To determine which classes have been registered, call *WinEnumObjectClasses*. To deregister a class, call *WinDeregisterObject*.

To replace a registered class with another class, call *WinReplaceObjectClass*. The replacement class will become a subclass of the one replaced. To remove the replacement class and restore the original class, call WinReplaceObjectClass again.

Object Instance Functions

To create an object instance, call *WinCreateObject* and specify the object's class name, title, setup, and location on the Desktop. All newly created objects return a persistent handle through WinCreateObject. This handle can be used at any time to reference the object.

If an application needs to receive a handle to an object that it did not create, call *WinQueryObject* and specify the full path name of the object or object ID. (The object ID is used to obtain a handle or pointer to the object.) Once the handle has been received by the application, call *WinSetObjectData* to change the state or behavior of the object. This function will call wpSetup to enable new KEYNAME values to be specified for the setup of the object.

Every WPS object class inherits a set of KEYNAMEs from the root class WPObject. To process an object's inherited KEYNAME information, call wpSetup when the

object is created. See "Using Setup Methods" on page 124 for more information on object setup.

To place an object in one of OS/2's predefined system folders, use one of the following IDs for the location:

<WP_CONFIG>	Place the object in the System Setup folder.
<WP_DESKTOP>	Place the object on the Desktop.
<WP_DRIVE>	Place the object in the Drives folder.
<WP_INFO>	Place the object in the Information folder.
<WP_NOWHERE>	Place the object in the Hidden folder.
<WP_START>	Place the object in the Startup folder.
<WP_SYSTEM>	Place the object in the OS/2 System folder.
<WP_TEMPS>	Place the object in the Templates folder.

To destroy an object instance, call WinDestroyObject.

Creating Help for WPS Objects

Help instances are created by the Workplace Shell using the *Information Presentation Facility (IPF)*. A help instance is associated with an object and supported by the *wpDisplayHelp, wpMenuItemHelpSelected*, and *wpQueryDefaultHelp* functions. See Chapter 10, "Creating Effective Help for Your Application."

Installing a Workplace Object

To install a WPS object, either run an installation program, run a batch file, or use the WPS Class List Object utility.

TIME SAVER

When you are in test mode (when you are compiling and testing, for example), place the following statement in your CONFIG.SYS file:

```
"SET OBJECTSNOOZETIME=0"
```

This will override the 90-second default and allow your object class DLL to be unloaded as quickly as possible when it is no longer in use by WPS.

Running an Installation Program

If an installation program is run, it should perform the following actions for the user:

1. Copy the DLL containing the object's class definition to the \OS2\DLL directory or to a directory in the LIBPATH.
2. Register the object class and the name of its DLL with the Workplace Shell by calling WinRegisterObjectClass.
3. (Optional) Create an object instance of the class and place it on the Desktop or in a folder by calling WinCreateObject.

When a class is registered, an object template is placed in the Templates folder (if the object class supports templating). Users can create object instances from a copy of the template (see "Using Object Templates" on page 131). An example of these processes is shown in Figure 7-8.

Figure 7-8. Registering an Object

```
fSuccess=WinRegisterObjectClass(pszClassName, /* Pointer to object class name */
                                pszModname)   /* Pointer to DLL name          */
if (! fSuccess)
{
    WinMessageBox(HWND_DESKTOP, hwnd,
                  "The call to WinRegisterObjectClass failed.",
                  "Add a new class", IDPB_OK, MB_OK | MB_ICONHAND);
}
```

Running REXX-Language Batch Files

An installation batch file can perform the same operations as an installation program by using the *REXX language utility functions* instead of the WPS functions (see Figure 7-9). REXX, which comes with OS/2 2.1, is a powerful and easy-to-write language that replaces the simple, batch-file language that comes with DOS.

Figure 7-9. Registering a Class with REXX

```
/* Load the REXX utility function */
call 'RxFuncAdd 'SysLoadFuncs', 'RexxUtil', 'SysLoadFuncs'
call SysLoadFuncs

/*Register a new object class with the Workplace Shell*/
if SysRegisterObjectClass("MyObjectClass", "MyDll") then
   say 'Class registration successfully completed for MyObject'

/* Create an object instance */
if SysCreateObject("MyObjectClass","A New Object","<WP_Desktop>",
                   "OBJECT=<A New Object>") then
   say 'A new Object has been created and placed on the Desktop'
```

Refer to the OS/2 2.0 Technical Library for more information on writing in the REXX language.

NOTE

If you plan to use REXX, you must install it using the OS/2 Setup and Installation screen. See Chapter 2, "Tips on Installing OS/2 2.1."

WPS Class List Object Utility

The *WPS Class List Object* utility provides a windowed user interface that is used for object creation and WPS class registration. WPS Class List is automatically installed during the installation of the OS/2 2.x Toolkit.

All registered classes are displayed in a hierarchical list by the WPS Class List object. The user can add, replace, or delete a class from the list by using the appropriate Win function (see "Using Objects in the Workplace Shell" on page 132).

TIP

The WPS Class List is an easy way to build and test objects for application development. However, because this tool can be used to delete any WPS object class (except for the predefined WPS classes), it is important to understand how to recover an object class if it is accidentally deleted. For this reason, we recommend that you *not* distribute the WPS Class List Object to your customers with your application. The preferred method of delivering objects to customers is through an installation program or batch file.

Migrating Applications

Migrating applications under OS/2 2.1 is as easy as installing them in their native mode. To add a DOS or Windows program to the Desktop, copy (click the right mouse button on the icon and select **Copy**) a DOS object from the Command Prompts folder to the Desktop or an appropriate folder. Click on the right mouse button to display the pop-up menu for the DOS object. Click on the arrow to the right of the **Open** option and then click on the **Settings** option to bring up the Settings notebook. Enter the name and path of the application on the Program page of the notebook. To add an OS/2

program to the Desktop, repeat this procedure but copy an OS/2 object instead of a DOS object.

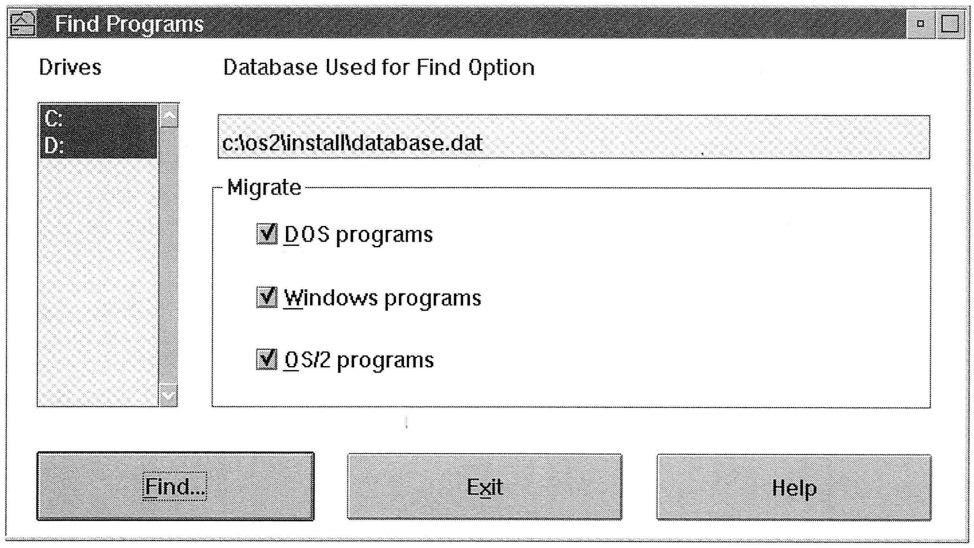
OS/2 2.1 provides a feature called *DOS Settings* to further customize your DOS or Windows applications. To get to this option, click on the **Sessions** tab in the Settings notebook to bring up the Sessions page and then click on the **DOS settings** option to select the settings you desire.

To assist you in application migration, OS/2 2.1 has set up a default database for many of the most popular DOS and Windows applications. This database is located at *\OS2\INSTALL\DATABASE.DAT*.

To select the default database from the Desktop:

1. Double-click on the OS/2 System object.
2. Double-click on the System Setup object.
3. Double-click on the Migrate Applications object to display the Find Programs screen (see Figure 7-10).

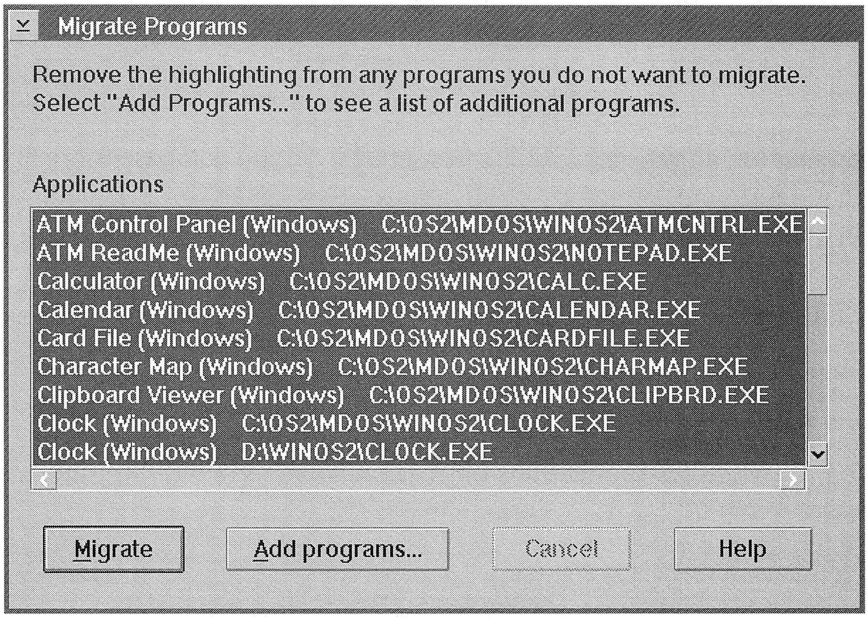
Figure 7-10. Find Programs Screen



4. Select the application types you want to migrate and the drives on which they are located.

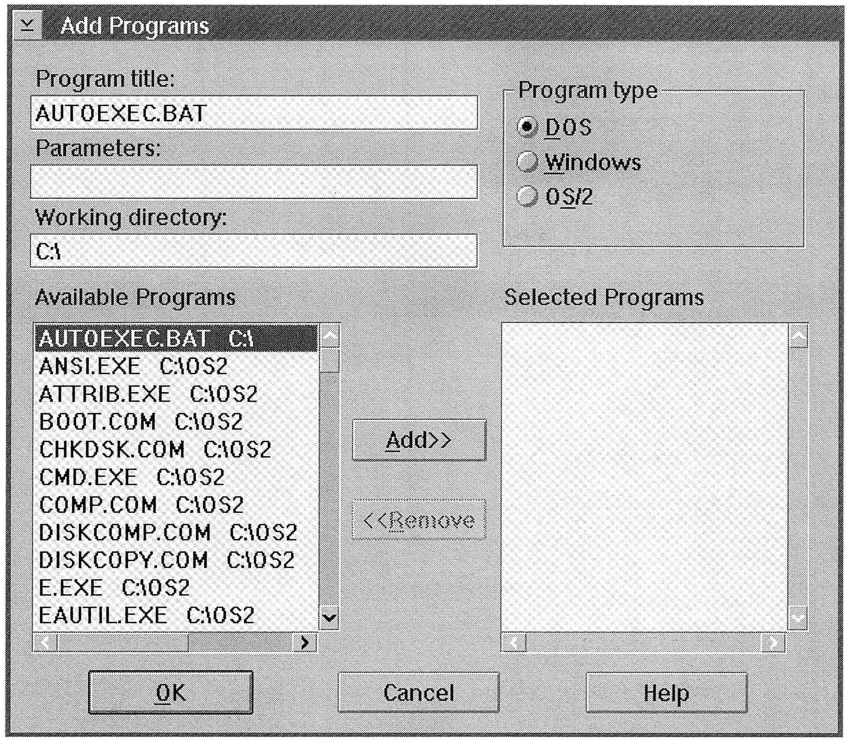
5. Click on the **Find** button to display the Migrate Programs screen (see Figure 7-11). The applications that are found will be listed and highlighted. (To deselect an application, simply click on it to remove the highlight.)

Figure 7-11. Migrate Programs Screen



6. If you are satisfied with your selections, click on the **Migrate** button to return you to the Find Programs screen.
7. If you do not see the application you are looking for, you can add it to the database by clicking on the **Add programs** button *before* clicking on the **Migrate** button. This will display the Add Programs screen (see Figure 7-12 on page 139).
8. Highlight the appropriate application and make any necessary changes to the title, parameters, or path.
9. Then click on the **Add** button to add the application to the Selected Programs window. Repeat steps 8 and 9 for each application you want to add.
10. After you have finished, click on **OK** to return to the Migrate Programs screen.
11. Click on the **Migrate** button to migrate the applications and return to the Find Programs screen.
12. Click on the **Exit** button to exit the Find Programs screen.

Figure 7-12. Add Programs Screen



OS/2 2.1 creates the following folders (which are placed on the Desktop) to contain your migrated applications:

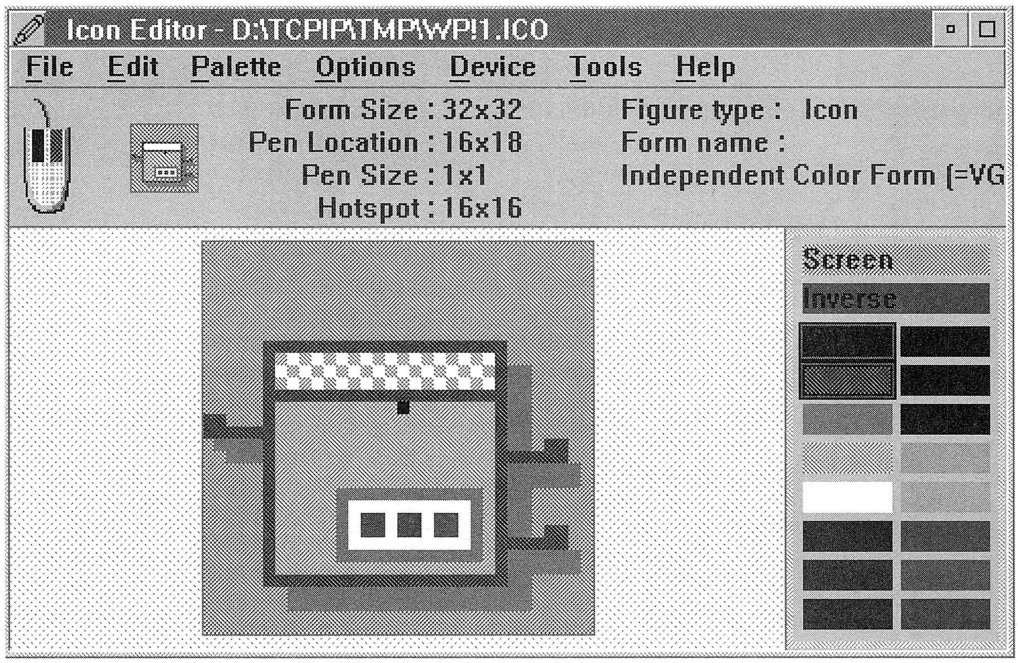
- DOS Programs
- Additional DOS Programs
- Windows Programs
- Additional Windows Programs
- OS/2 Programs
- Additional OS/2 Programs
- Win-OS/2 Groups

Setting Up New Icons for Your Applications

To set up a new icon for an application, copy an existing icon and modify it using a tool provided by OS/2 2.1, such as the Icon Editor (see Figure 7-13 on page 140). Icons can be moved out of one folder and into another folder or subfolder or onto the Desktop by dragging and dropping them.

Applications can be set up to execute automatically when the system starts or when the user double-clicks on an icon.

Figure 7-13. Icon Editor



Chapter 8.
Communicating among Machines Using TCP/IP

For the OS/2 operating system to become a part of the corporate world, it must be able to communicate with other machines. Few companies have a single network with one type of machine running a single operating system. To be effective, a personal computer that uses OS/2 2.1 needs to be able to work in a heterogeneous network environment and communicate with a variety of machines such as a workstation running UNIX, a mainframe running VM/CMS, or another personal computer running DOS. One way to do this is to use the *Transmission Control Protocol/Internet Protocol (TCP/IP)*.

Some of the other network protocols and products that are used in corporate computing are listed in Table 8-1. We have concentrated on TCP/IP because it is generally acknowledged as the common denominator in network communications.

Table 8-1. Other Network Protocols

Network	Company
NetWare	Novell, Inc.
Network Basic Input/Output System (NetBIOS)	Microsoft Corporation
System Network Architecture (SNA)	IBM Corporation
X.25	Consultative Committee International Telephony and Telegraphy

What Is TCP/IP?

In recent years, TCP/IP has emerged as the most common communication protocol. If a machine must communicate with another machine of a different type, structure, or operating system, the only currently assured means is through TCP/IP. Most machines have some implementation of TCP/IP available, making standardized communication across varied architectures and operating systems possible. TCP/IP runs on various platforms, including IBM/370 machines running VM, SUN workstations running SUN/OS, Apple Macintosh machines, and DEC VAX clusters.

TCP/IP enables file transfer between machines, remote execution of programs, electronic mail transfer, remote logon, booting of a diskless machine off a remote server,

client-server application communication, drive and file-system sharing, electronic news reading, and more.

Strictly speaking, TCP/IP is a *protocol suite*, which is a group of interrelated protocols. TCP/IP can be thought of on three levels:

- Protocol specification, which is the actual format definition of the frames and datagrams used by TCP/IP developers and low-level network software.
- System function calls, which are the interface between the application programs and the network. These functions are used by TCP/IP application programmers.
- Application programs, which are the programs that make TCP/IP useful (that is, they are the ones used by end users).

Applications tend to vary more than system calls so they can take advantage of unique features of the operating system. However, the protocol suite is standardized. Any machine should be able to talk to any other machine as long as it obeys the standard. The formal specifications of the protocol suite are in the *Request For Comment (RFC)* documents, which are maintained by SRI International, also known as the Network Information Center. Because of their complexity and unwieldiness, they are rarely used as references in application development. The primary source for TCP/IP system call documentation is the *IBM TCP/IP Programmer's Reference*, which comes with the TCP/IP Programmer's Toolkit. Although TCP/IP is generally considered to be standardized, system calls do vary slightly with implementation. The examples we use in this chapter refer to the IBM TCP/IP Version 1.21 for OS/2 2.x.

Fundamental Protocols of TCP/IP

TCP/IP is built upon three fundamental protocols: the Internet Protocol, User Datagram Protocol, and the Transmission Control Protocol. All TCP/IP protocols and application programs use one or more of these base protocols.

The *Internet Protocol (IP)* datagram is the basic unit of transmission. A *datagram* is the smallest data packet of IP information; it is a connectionless, best-effort service. This means there is no guarantee that it will reach the other host. Because it is unreliable, other protocols are necessary. In addition, it does not specify a port on the destination machine, which makes IP less than useful for meaningful communication.

The *User Datagram Protocol (UDP)* is built upon IP and includes a port number, which enables application programs on different machines to communicate, and (usually) a *checksum*, which is an integer value used to determine the integrity of a group of data. However, it does not guarantee delivery.

The *Transmission Control Protocol (TCP)* is also built upon IP and is the standard protocol most applications use. It provides a reliable, full-duplex, stream service. It is also connection oriented, which means that programs must establish a connection before they can transmit data.

Other TCP/IP Protocols and Applications

Most TCP/IP implementations include the protocols/applications listed in Table 8-2.

Table 8-2. TCP/IP Protocols and Applications

Protocol	Name	Description
BOOTP	Bootstrap Protocol	Enables a diskless machine to boot off a remote boot server.
FTP	File Transfer Protocol	Exchanges files between hosts.
NFS	Network File System	Mounts remote file systems so they can be accessed as if they were local.
REXEC	Remote Execution	Runs a program on a remote host.
SMTP	Simple Mail Transfer Protocol	Exchanges mail messages with remote hosts.
SNMP	Simple Network Management Protocol	Reports network errors and management information to the source.
TELNET	Telnet	Logs onto a remote host.

Internet Addressing and Domains

For one host to communicate with another host, they must both abide by the internet addressing scheme. One way to identify a machine is with an IP or *internet address*. An IP address is 32-bits long. There are five types of addresses:

Class A For extremely large networks. There can be only 128 of these networks, but they can have between 2^{16} (65,536) and 2^{24} (approximately 16 million) hosts.

Class B For intermediate-sized networks. There can be 2^{14} (16,384) of these networks; they can have between 2^{8} (256) and 2^{16} (65,536) hosts.

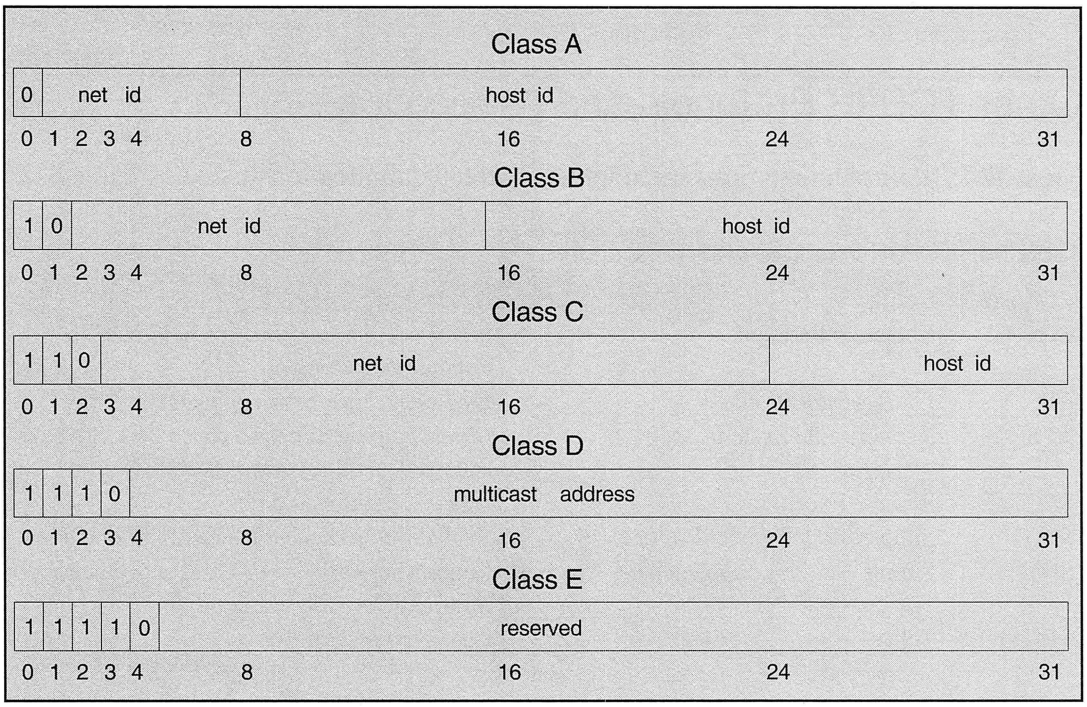
Class C For small networks. There can be 2^{21} (approximately two million) of these networks, but they can only have up to 256 hosts.

Class D For multicast addressing (not commonly used).

Class E Reserved for future use.

The format for the address types is shown in Figure 8-1.

Figure 8-1. Address Type Formats



Most people are more accustomed to decimal numbers than to binary numbers. For this reason, the IP address is often represented in dotted, decimal format. Each byte is grouped as a unit of binary digits separated by a period. For example, the IP address *10001001 01010011 00000001 00101111* is represented as *137.83.1.47*.

The IP address has another representation: the host name and the domain name pair. The *host name* is the name of the particular machine; the *domain name* is a local, group, and site subdivision of the network. For example, a host name and domain name combination such as ELVIS.GRACELAND.MEMPHIS.COM would be host name ELVIS, local subdivision GRACELAND, group MEMPHIS, and site COM.

This example is a Class B address type because the first two bits of the first grouping are 10 (therefore, ELVIS is in a medium-sized network).

Nameservers and Routers

Because it is more convenient for you to refer to a machine by a name such as ELVIS.GRACELAND.MEMPHIS.COM than by a number such as 9.99.1.47, it is necessary to have a mechanism for translating the name to the address. On small networks this can be accomplished by creating a HOSTS file that contains all the host names and IP addresses of every machine on the network. However, as the network grows, this quickly becomes unmanageable. A better solution is to centralize the translation process through the use of a *nameserver*, which is a machine that translates host names to IP addresses and vice versa.

For a machine on one network to be able to communicate with a machine on an adjoining network, a *router* must connect the two networks. This is a machine that knows the address range of both networks and when to route packets from one network to the other.

Using TCP/IP in Your Application

The two main ways of using TCP/IP in your application are through socket subroutines and remote procedure calls.

> **NOTE**
>
> Because the IBM TCP/IP Version 1.21 for OS/2 is currently built on 16-bit libraries, use of these TCP/IP APIs forces you to use mixed-mode programming. Care must be taken to ensure the proper callback so that your 32-bit code provides a 16-bit entry point. The use of Seg16 tells the compiler to *thunk,* or convert from a 16:16 pointer to a 0:32 pointer. (The new release of IBM TCP/IP for OS/2 is a full 32-bit implementation that makes mixed-mode programming unnecessary.)

Sockets

A *socket* is an endpoint for communication that can be addressed by the network; it is created so that the TCP/IP programs can communicate between hosts. Sockets are

duplex (bi-directional). Once the socket is created, data can be sent and received simultaneously. Sockets are represented as socket descriptors (integers).

The three classes of sockets are stream, datagram, and raw. They are described as follows:

- *Stream sockets*, which are based on TCP, are connection-oriented and divided into a server socket and client socket. When the server socket is established, it waits for a client application to connect. The client application initiates the connection and can then communicate with the server until the socket is closed.
- *Datagram sockets*, which are based on UDP, consist of connected and unconnected sockets. Connected sockets communicate with only one host; unconnected sockets communicate with any host. Because UDP is unreliable, packets sent through these sockets are not guaranteed to arrive.
- *Raw sockets* allow access to low-level protocols such as IP or the Internet Control Message Protocol (ICMP). These sockets are rarely used in application programs.

TIP

IBM TCP/IP for OS/2 socket implementation differs from the Berkeley socket implementation. OS/2 sockets are not files or devices; they cannot be accessed by calling the standard *read()*, *write()*, or *close()* functions. Instead, you should use the *recv()*, *send()*, and *soclose()* functions. Also, you should use the include file *<nerror.h>* instead of *<errno.h>* for reference to errors, and the *tcperrno()* function instead of *errno()* for return values.

A stream socket example of a client process is shown in Figure 8-2 on page 147. In this example, an OS/2 machine queries a server for its load level to determine if the server is busy. A server process, or *daemon,* runs on the central machine and is the server end of the socket. The OS/2 machine runs the client process to query the server and the daemon responds with its current load level.

Remote Procedure Calls

Remote procedure calls (RPC) permit remote execution of a subroutine across a TCP/IP network. The server side supplies a program that consists of a set of service procedures. The combination of host address, program number, and procedure number define a remote service procedure. The client application sends a packet to the server

Figure 8-2. Example of a Stream-Socket Client Process

```
/*To begin every socket program, you must execute the sock_init() function to */
/*initialize sockets. If sock_init() fails, TCP/IP could be having problems.  */

rc = sock_init();
if(rc!=0)
{
   fprintf(stderr,"error: sock_init()failed, tcp/ip not working\n");
   exit(2);
}

/*The gethostbyname() function is called to obtain the host information, and */
/*then bcopy() is called to set up the sockaddr_in structure.                */

host_pointer = gethostbyname(hostname);
if{host_pointer==NULL)
{
   fprintf(stderr,"error: cannot find %s\n",hostname);
   exit(1);
}

bcopy(host_pointer -> h_addr,
      (caddr_t)&server_addr.sin_addr,
      host_pointer -> h_length);

/*The socket is created with a call to the socket() function. The domain      */
/*(AF_INET), the type (SOCK_STREAM), and the protocol (0) is passed to it. A  */
/*value of 0 for the protocol tells the system to use the default protocol    */
/*type for the domain and type.                                               */

sock_num = socket(AF_INET, SOCK_STREAM, 0);
if(sock_num<0)
{
   fprintf(stderr,"error: socket()call failed\n");
   exit(2);
}

/*Once the socket is created, the sin_family and sin_port structure elements */
/*can be initialized and the socket can connect to the server.               */

server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(LEVPORT);
rc = connect(sock_num,(char*_Seg16)&server_addr,(short)sizeof(server_addr));
if(rc<0)
{
   fprintf(stderr,"error:cannot connect to %s",hostname);soclose(sock_num);
   exit(1);
}

/*To translate LEVPORT into network byte order, call the htons() function. (In */
/*RPC, the XDR filters take care of this.) The connect() function is passed the*/
/*socket number to be connected, a pointer to the socket_addr structure, and   */
/*the size of the sock_addr structure. The output buffer is filled with the    */
/*data, the data size is initialized, and the data is then sent.               */

strcpy(outbuff, LEVREQ);
bytes_to_send = strlen(outbuff);
bytes_sent = send(sock_num, outbuff, bytes_to_send, 0);
if(bytes_sent < bytes_to_send)
{
   fprintf(stderr,"error: send() failed\n");
   exit(1);
}
```

Example of a Stream Socket (continued)

```
/*The send() function returns the number of bytes actually sent; this should */
/*match the number that was attempted to be sent. The response is received   */
/*from the server by calling the recv() function, which is similar to send().*/

bytes_recv = recv(sock_num, inbuff, BUFFLEN, 0);
if(bytes_recv < 0)
{
   fprintf(stderr,"error: recv() failed\n");
   exit(1);
}

/*BUFFLEN is the size of the receive buffer. If this were a datagram socket  */
/*and the data to be received was larger than the buffer, the excess would be*/
/*discarded. Because it is a stream socket, the server process waits until   */
/*another recv() is called in case there is excess data. After all the data  */
/*is read, a call to the soclose() function finishes the transaction.        */

rc = soclose(sock_num);
if (rc != 0)
{
   fprintf(stderr,"error: soclose() failed\n");
   exit(2);
}

/*Results of the exchange are placed in the inbuff field and are ready for use.*/
```

defining the service requested. The server then performs the requested service and sends a reply back to the client.

There are three levels of RPC: high, intermediate, and low. Choosing which RPC level to use is a matter of deciding what needs to be done. Start at the top level, then work down to a level where the data transfer can be accomplished.

In *high-level RPC*, all the protocol details are hidden and the RPC is totally transparent to the program. Your application simply makes a call to a front-end routine, which does all the communication work and returns the information. This is not actually a part of RPC; it is just a system call with the host name.

At the *intermediate level*, you have moderate control (see Figure 8-3). RPCs are made with the system routines *registerrpc(), callrpc()*, and *src_run()*. Most applications use this layer because of its balance of control and complexity.

Low-level RPC allows you complete control over the details of the RPC (see Figure 8-4 on page 150). It must be used when long data streams or server authentication is required. Low-level RPC requires familiarity with sockets and their system function calls.

Figure 8-3. Example of an Intermediate-Level RPC Call

```
/*This RPC performs a simple, remote-system, information query. It returns a */
/*value indicating whether the remote system has a disk or is diskless.      */

rc = callrpc(hostname,
             RSTATPROG,
             RSTATVERS,
             RSTATPROC_HAVEDISK,
             xdr_void,
             0,
             xdr_u_long,
             (char *)&hasdisk);
if(rc != 0)
{
   fprintf(stderr,"error: could not get havedisk\n");
   return(2);
}

/*Note that xdr_u_long is a function call that is passed as an argument to the*/
/*callrpc() function so that callrpc() can translat the data correctly.       */
/*RSTATPROG, RSTATVERS, and RSTATPROC_HAVEDISK are the RPC constants that     */
/*define which RPC to execute on the remote system. The parameters,xdr_void   */
/*and xdr_u_long, are XDR Integer Filter Primitives (xdr_void is used because */
/*this remote procedure does not take any arguments;xdr_u_long is used because*/
/*the remote procedure returns an unsigned LONG, which is typecast to CHAR).  */
/*The value hasdisk is used to determine whether the system has a disk or not.*/

if ((int)hasdisk > 0)
{
   printf("host %s has a disk\n", hostname);
}
else
{
   printf("host %s is diskless\n", hostname);
}
```

In the socket example (Figure 8-2 on page 147) and the low-level RPC example (Figure 8-4 on page 150), a similarity of system functions and data structures is evident. This is because almost every RPC implementation uses sockets at the lowest level. RPCs are actually a special case of socket communication.

TIP

RPCs are often preferred over sockets because they can take some of the work out of building an application. If your application fits into the RPC format, consider using RPCs because of their simplicity. However, if you need extra control, then sockets are your best choice.

RPC uses External Data Representation (XDR), which translates data so it can be transferred to different machines. Sockets have no mechanism for this; your application must either handle the translation or use RPC.

The examples in Figures 8-2, 8-3, and 8-4 are all from the client application's point of view. Server examples have been excluded because of space limitations and the fact that they are less common under OS/2 2.1. However, TCP/IP server communication applications are similar to client applications with only slight modifications.

Figure 8-4. Example of a Low-Level RPC Call

```
host_pointer = gethostbyname(hostname);

if(host_pointer==NULL)
{
   fprintf(stderr, "error:cannot find %s", hostname);
   return(1);
}

/*The host_pointer variable returned by gethostbyname is a pointer to a      */
/*hostent structure, which is defined in the include file <netdb.h>. The     */
/*structure element that is needed from this call is the host address. A call*/
/*to bcopy() fills in the sin_addr field of server_addr.                      */

bcopy(host_pointer -> h_addr,(caddr_t)&server_addr.sin_addr,
      host_pointer -> h_length);

/*The sin_family and sin_port elements of the server_addr structure are      */
/*initialized, the time-out values are set, and the client transport handle  */
/*created for the remote program.                                             */

server_addr.sin_family = AF_INET;
server_addr.sin_port   = 0;
retry_timeout.tv_sec   = 4;
retry_timeout.tv_usec  = 0;
client_pointer = clntudp_create(&server_addr, RSTATPROG, RSTATVERS,
                                (retry_timeout, short *)&sock);

if (client_pointer == NULL)
{
   clnt_pcreateerror("clntudp_create");
   return(2);
}

/*AF_INET refers to internet domain addressing, which is currently the only    */
/*addressing supported in the IBM TCP/IP. The retry_timeout parameter indicates*/
/*the time-out value between tries. (This is only necessary for UDP transport  */
/*handles; for TCP handles, a send and receive buffer is defined instead.) Once*/
/*the handle is created, the actual RPC is invoked with a call to clnt_call(), */
/*which contains information about the client, remote procedure, XDR filters,  */
/*and timeout. It returns the output value from the remote procedure.          */

total_timeout.tv_sec = 45;
total_timeout.tv_usec = 0;
client_stat=clnt_call(client_pointer, RSTATPROC_HAVEDISK, xdr_void, 0, xdr_u_long,
                      (char*)&hasdisk, total_timeout);

if(client_stat!= RPC_SUCCESS)
{
   clnt_perror(client_pointer,"rpc");
   return(2);
}
/*The results of the clnt_call() function is in hasdisk, which now indicates */
/*whether or not the remote machine has a disk.                               */
```

Using External Data Representation

External Data Representation (XDR) is a data representation standard that allows different machines running different operating systems to exchange data; it is independent of the hardware and the operating system. By using the XDR filters and subroutines, a program can be assured of being understood by any other program using XDR. For example, to pass an unsigned LONG integer from one machine to another, the *xdr_u_long* filter is used to translate the value into XDR. The process running on the other machine translates the value from XDR into its local representation of an unsigned LONG integer. In this manner, the two machines can communicate without each one knowing how the other one stores data. An example of this is shown in the RPC code in Figure 8-3 on page 149.

If a RPC needs to use the Transmission Control Protocol (TCP) for the transport protocol, it must use the low-level layer. (Both upper-level layers use only the User Datagram Protocol.)

The low-level RPC (Figure 8-4) achieves the same results as the intermediate-level RPC example (Figure 8-3). For the low-level RPC, however, more information about the host is needed. This information is obtained with a call to *gethostbyname()*. Note that the only differences between *clnt_call()* in this example and *callrpc()* in the intermediate example are the use of client structure from *clntudp_create()* and the definition of the timeout value.

The result of the RPC is in *hasdisk*. By comparing the intermediate- and low-level examples, you can see that the trade-off for the extra flexibility of the low-level example is a moderate increase in complexity.

Less Common Protocols

Because TCP/IP is continually evolving, new protocols are constantly being added and protocols that are no longer useful are being deleted. For example, some of the protocols under development are the Network Voice Protocol (for transfer of voice data) and the Network Video Protocol (for transfer of video data). Protocols that are no longer in common use include Remote Job Entry (RJE), Network Standard Text Editor (NETED), and Cross Net Debugger (XNET).

Some implementations of TCP/IP include less common protocols such as the ones shown in Table 8-3.

Table 8-3. Other TCP/IP Protocols

Protocol	Name	Description
NNTP	Network News Transfer Protocol	Transfers news articles from one machine to another.
NTP	Network Time Protocol	Synchronizes all machines on the network to the correct (or common) time.
PPP	Point to Point Protocol	Runs TCP/IP applications over a phone line or serial link. This has superseded SLIP as a more efficient protocol.
SLIP	Serial Line Internet Protocol	Runs TCP/IP applications over a phone line or serial link.

Setting Up the Development Environment

To compile and link TCP/IP programs, the TCP/IP directories should be added to the following environment variables:

```
INCLUDE=c:\tcpip\include;
LIB=c:\tcpip\lib;
```

The compile line will look similar to the following:

```
FNAME=yourprog
COPTS= /C /L /Ls /Li /Kb /W3 /Ti /Se /Ss /Gs /Gt /Ms
icc $(COPTS) $(FNAME).c
```

The link line will look similar to the following:

```
LOPTS= /BASE:65536 /DEBUG /NOI /MAP
STDLIBS=DDE4SBS+OS2386+DDE4SBM
TCPLIBS=tcpipdll+rpcdll
FNAME=yourprog
link386 $(LOPTS) $(FNAME), $(FNAME).exe,,
$(STDLIBS)+$(TCPLIBS),$(FNAME).def;
```

Part III. Implementing Programming Solutions

Chapter 9.
Programming with the Modular Method

Presentation Manager (PM) is an event-oriented programming environment that provides a user-friendly front end for OS/2. Event-oriented applications respond to messages posted to a message queue. A specific message determines the action to be taken, such as data initialization, window painting, window destruction, etc. There is usually more than one message queue in a multitasking system because each separate thread or process requires a specific message queue.

Presentation Manager is well-suited for a modular approach to designing applications. Typically, OS/2 and Presentation Manager functions are imbedded in the code. However, using *function wrappers* in a modular design allows you to repeatedly use the same code, which saves time and reduces maintenance. Modular coding also speeds up the development of new applications and reduces your learning curve.

Using a Modular Approach to Coding

This section begins the process of moving away from coding with the Case statement method (generally known as "the Case Statement from Hell!") and moving towards a more modular approach to coding. This means applying the most commonly used OS/2 functions to small code modules to quickly create a usable application that runs under OS/2 2.1.

By using the techniques provided in this chapter, you can avoid the problems most programmers have when learning OS/2 2.1. Some of the more typical problems are mistyping a common PM or OS/2 function call, mixing up resource identifiers, making extensive code changes instead of minor ones, and poring over unreadable code in the wee hours of the night with a deadline coming up.

The code discussed in this chapter (and found on the companion diskette) contains functions that can be copied verbatim. If you want to combine some of the functions in a different manner, go ahead and experiment. You will find that the modules are quite flexible and can adapt to most of your programming needs.

Using Function Modules

Because the applications you design have commercial uses, your required functions should include all the system functions. Although this means you will be using two functions instead of one, the functions you build with the modular approach will make the code cleaner and, therefore, easier to read.

System functions have been known to change. For example, some of the OS/2 2.1 functions now have fewer parameters than the same functions used in OS/2 1.3. Others have been renamed. Although this does not happen often, we recommend that you practice *defensive programming* and protect your applications from this possibility by using the modular approach. It is a lot less work to change two or three functions and relink them than it is to search through all your code to change a large number of system functions.

Types of Function Models

There are several methods of approaching individual functions:

- Link each separately compiled function.
- Build a library of functions.
- Put all the functions into a dynamic link library (DLL).

Building a library means trading size for efficiency, whereas building a DLL means all tasks can share the same code. A DLL also ensures that all functions are truly reentrant.

TIP

Most standard *C* libraries are not reentrant; therefore, we recommend you use the supplied OS/2 functions (for example, *DosAllocMem* for *malloc*).

To build a new application, compile each function separately and let the make file take care of rebuilding the entire application. These separate functions will become part of a common pool of functions that you will be able to use in other applications. This may seem strange or cumbersome at first, but it is actually more efficient. Linking one or two submodules is always faster than compiling and linking one large module.

Using a Magic Window

When writing graphical user interfaces, programmers generally use a test application called a *Magic Window*. The Magic Window is used as a prototype for testing and experimenting. It is also useful for introducing new controls and the functions that created them. Table 9-1 shows some of the reusable functions that we used to create our Magic Window.

Table 9-1. Magic Window Functions

Function	Module	See Page
CreateContainer	OS2F0010.C	368
CreateNotebook	OS2F0006.C	365
CreateWindow	OS2F0004.C	363
DisplayListBox	OS2F0008.C	367
GetSelectedLBEntry	OS2F0003.C	363
InitEntryField	OS2F0011.C	369
InsertNotebookPage	OS2F0007.C	366
IsCtlButtonOn	OS2F0013.C	370
MagicWndProc	MAGICWP1.C	168
Main	MAGICW.C	159
ToggleButtons	OS2F0012.C	369

The source modules that begin with *MAGIC* were specifically created for this program; however, they also can be used as templates for other applications. All modules beginning with *OS2F* are reusable functions that perform one specific task (see Appendix A, "Reusable Submodules".

The source code for the Magic Window application is shown in the mainline template. Use the make file in Figure 9-4 on page 166 to create this Magic Window. See Chapter 3 for information on how to set up your resource and definition files.

Creating the Mainline Template

A *mainline* template is useful when creating Presentation Manager (PM) applications for the OS/2 operating system. This template usually contains the following steps, which must be performed by all PM applications:

1. Initialize the application. This registers the program with the operating system and returns a unique *anchor block* handle.

2. Create the main message queue.
3. Instantitate the Help system if you will be adding Help to your application. See Chapter 10, "Creating Effective Help for Your Application".
4. Register the classes of all the windows you will be using.
5. Perform application-specific functions such as creating windows, creating list boxes, opening databases, etc.
6. Add the application name to the Task Manager list, if desired. This will allow the application to be selected from the Window List, which is displayed when the Ctrl+Esc keys are pressed.
7. Enter the main message processing loop; read and process all messages until the WM_QUIT message is received; then exit the loop.
8. Destroy the Help facilities if they were originally instantiated.
9. Destroy all windows. (Note that destroying the primary window will automatically destroy all secondary windows associated with it.)
10. Destroy the message queue.
11. Terminate the application and disassociate it from Presentation Manager.

The code used to accomplish these steps is shown in Figure 9-1. It can be used with little or no change for every application you write for OS/2 2.1.

To tailor this code for your own use, change or add the following items:

1. Change the sample application name *MAGICW* to the application name that you want to register.
2. Change the sample main window procedure name *MagicWndProc* to the main window procedure name that you want to register.
3. Change the identifiers (*ID_MAGIC* and *IDS_MENU_TITLE* are used in the sample).
4. Change the frame window title in the Magic Window sample application to the title of your application.
5. Add any application-specific functions, if needed.
6. Tailor the help instance data (particularly the help library name) and the Help window title.

Figure 9-1. Submodule MAGICW.C

```
/*============================== MAGICW.C ====================================
FUNCTION:    Main
DESCRIPTION: Main module for the Magic Window
PARAMETERS:  int argc, char *argv[]
RETURNS:     INT
EXPORTS:     N/A
============================================================================*/
#define INCL_PM

#include <os2.h>
#include <string.h>
#include "menures.h"
#include "magicwp1.h"
#include "msghndlr.h"
#include "syshelp.h"

HAB hAB;
HWND hWndMain;

int main(void)
{
   QMSG   qmsg;                    /* MSG structure to store messages          */
   HMQ    hMQ;                     /* Message queue handle                     */
   HWND   hWndClient;              /* Client window handle                     */
   CHAR   szTitle[30];             /* Load title string into here              */
   ULONG  flCreate = FCF_STANDARD; /* Use standard frame control flags         */

   if((hAB = WinInitialize(0L)) == 0L)
   {
      return(FALSE);
   }
   if((hMQ = WinCreateMsgQueue(hAB, 0L)) == 0L)
   {
      return(FALSE);
   }
   if(!WinRegisterClass(hAB, "MagicClass", MagicWndProc, CS_SIZEREDRAW, 0))
   {
      return(FALSE);
   }
   WinLoadString(hAB, 0L, IDS_MENU_TITLE, sizeof(szTitle), szTitle);

   hWndMain = WinCreateStdWindow(HWND_DESKTOP,     /* Desktop window is parent */
                                 WS_VISIBLE,       /* Frame style              */
                                 &flCreate,        /* Frame control flag       */
                                 "MagicClass",     /* Window class name        */
                                 NULL,             /* Window title bar         */
                                 0L,               /* Client style of VISIBLE  */
                                 (HMODULE)NULL,    /* Resource is in .EXE file */
                                 ID_MAGIC,         /* Frame window identifier  */
                                 &hWndClient);     /* Client window handle     */

   if(hWndMain == 0L)
   {
      return(FALSE);
   }

   /* Set the title bar because it wasn't set when the window was created.     */

   WinSetWindowText(hWndMain, szTitle);
   HlpOpenHelp(hWndMain,0L);                                 /* Initialize help */
   WinSendMsg(hWndClient, WM_PAINT, 0, 0);
```

Submodule MAGICW.C (continued)

```
   /* Switchlist structure removed for adding to Window List. */
   /* The following is the message loop for the application.  */

   while(WinGetMsg(hAB, (PQMSG)&qmsg, 0L, 0, 0))
   {
      WinDispatchMsg(hAB, (PQMSG)&qmsg);
   }

   HlpDestroyHelp();

/* Perform clean up before exiting application. The following routine          */
/* destroys the application's frame window (which also destroys its child     */
/* windows), destroys its message queue, and disassociates the application    */
/* from the Presentation Manager system.                                      */

   WinDestroyWindow(hWndMain);        /* Destroy the frame window             */
   WinDestroyMsgQueue(hMQ);           /* Destroy this application's message clue */
   WinTerminate(hAB);                 /* Terminate this application's use of the */
                                      /* Presentation Manager resources       */
   return(0);
}
```

If you provide identifier definitions in the header file *MAIN.H*, the code in Figure 9-1 can be compiled as a separate object module, which can be linked with your other window and dialog procedure modules.

Window and Dialog Procedure Styles

Because window and dialog procedures process only messages, they usually consist of a large "switch" statement with case statements for each message on which the procedure takes action. Although the execution of an application depends upon the messages received, coding the case statements in-line does not lead to an easy understanding of what the application does. In addition, writing all the case statements usually requires several *pages* of code.

We recommend a different approach: condense the case statements and create functions for the messages and procedural code. (The procedural code executes as a consequence of trapping a specific message.) Although this is not as easy as the usual case statement method because it requires more thought and planning, it results in templates and functions that can be reused over and over with little or no alteration.

The code for a typical main window procedure is shown in both Examples A and B (see Figures 9-2 and 9-3 on page 162).

Figure 9-2. Example A. Traditional Window Procedure

```
MRESULT EXPENTRY MyWindowProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
   switch(msg)
   {
      case WM_CREATE:
         /*Required initialization code(when a window is created)goes here.*/
         break;
      case WM_COMMAND:
         /* Extract command and respond to user actions such as menu       */
         /* pull-downs, keyboard entry, etc.                               */
         command = SHORT1FROMMP(mp1);
         switch (command)
         {
            case ID_OPTION1:                  /*Perform action if selected.*/
               break;
            case ID_OPTION2:                  /*Perform action if selected.*/
               break;
            case ID_EXITPROG:
               WinPostMsg(hwnd, WM_CLOSE, 0L, 0L);
               break;
            default:
               return WinDefWindowProc(hwnd, msg, mp1, mp2);
         }
         break;
      case WM_PAINT:                          /*Draw window contents       */
         break;
      case WM_CLOSE:
         /*Execute any termination routines required, then post the       */
         /*termination message to the message queue.                      */
         WinPostMsg(hwnd, WM_QUIT, 0L, 0L);
         break;

      default:         /*All other applicable functions go here.           */
         return WinDefWindowProc(hwnd, msg, mp1, mp2);
         /*This call must exist in the window procedure!                  */
    }
    return FALSE;
}
```

These examples show the difference between the traditional approach to coding and the modular approach. Although there are special cases where a function must return TRUE to indicate that a specific message was processed, functions should normally return FALSE as shown in these procedures.

Both examples illustrate the point that *every* procedure must have a default procedure to handle any conditions that are not handled by your application. Notice, however, that in the modular approach:

- Not all of the message functions require that all four parameters be passed to the procedure.
- Because each message function stands alone, the procedure and each message function can be compiled and debugged separately.

Figure 9-3. Example B. Modular Window Procedure

```
MRESULT EXPENTRY MyWindowProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
   switch(message)
   {
      case WM_CREATE:
         return WMCreateMsg(hwnd);
      case WM_DESTROY:
         return WMDestroyMsg(hwnd);
      case WM_COMMAND:
         /* All required message functions go here.*/
         return WMCommandMsg(hwnd, message, mp1, mp2);
      case WM_CONTROL:
         return WMControlMsg(hwnd, mp1);
      case WM_CLOSE:
         WinPostMsg(hwnd, WM_QUIT, 0L, 0L);
         break;
      case WM_PAINT:
         return WMPaintMsg(hwnd);
      default:
         return WinDefWindowProc(hwnd, message, mp1, mp2);
   }
   return FALSE;
}
```

Designing your first PM application can be a slow process. However, the modules we have provided for you will save you considerable time and effort. After the first application design is completed, the rest should be easy!

Communicating with PM Windows

Windows and dialogs are created for the specific purpose of displaying or manipulating data for your application. They are destroyed after accomplishing their purpose.

The following actions are common means of communicating with and/or passing data to Presentation Manager windows:

- Send a message to the window and trap the message in the procedure.
- Pass the data as parameters by calling WinCreateMsg.
- Attach the data to a data area unique to the window by calling the WinSetWindow functions.

To communicate with any window, you must know its handle. Many systems use global data to create their handles. The advantage of this is that once the handle is obtained, it can be used by anyone. The disadvantage is that it also can be destroyed by anyone.

If you know a window's handle, use the following function to determine a window's identifier:

```
idChild = WinQueryWindow(hWnd, QWS_ID);
```

If you know the primary window's handle, you can obtain the application's anchor block by using the following function:

```
hAB = WinQueryAnchorBlock(hWnd);
```

If you know the primary window's handle and the identifier of an associated secondary window, you can determine the secondary window's handle as shown below:

```
hWnd = WinWindowFromID(hWndParent, idChild);
```

To obtain the handle of a specialized (system) window such as the system menu bar, use the preallocated system identifier:

```
hWnd = WinWindowFromID(hWndFrame, FID_SYSMENU);
```

These variables and functions are used in the samples in Chapter 11, "The Warehouse Inventory Application".

Sending a Message to a Window

Sending a message to a window is easy if you know its handle. Look up the message to determine what parameters need to be passed (if any). For example, the Paint message shown below does not require any parameters to be passed:

```
WinSendMsg(hWndClient, WM_PAINT, 0L, 0L);
```

A command message would require parameters similar to the following:

```
WinSendMsg(hWndDialog, WM_COMMAND, MPFROMSHORT(DID_CANCEL), 0L);
```

In this case, the handle was obtained by calling WinLoadDlg (the above code sends a message to cancel the call).

There are many times when processing must be performed in a window procedure but no standard message is available to do it. When this occurs, use a *user-defined* message. The system-defined identifier *WM_USER* is provided for this purpose.

Although WM_USER can be used as a starting ID, it is better to start with a higher increment (WM_USER +1) as shown below:

```
#define MY_MSG_1 (WM_USER + 1)
#define MY_MSG_2 (WM_USER + 2)
```

This way, if the client window procedure needs to send a message to another window procedure, it performs the following function. Note that the parameter is not necessarily required; it is shown here as an example.

```
WinSendMsg(hWndWinOther,MY_MSG_1,MPFROMLONG(mydata),0L);
```

The following code will trap the message in the window procedure that receives it and do any required processing:

```
case MY_MSG_1:
   /* Extract parameter with LONGFROMMP(mp1);process message */
   mydata = LONGFROMMP(mp1);
   break;
```

Passing the Data as Parameters

As explained previously, data can be passed as parameters by calling WinSendMsg or WinPostMsg; however, there are other methods that can be used. For example, to pass a simple, null-terminated string to a dialog procedure, you can use the following function:

```
WinDlgBox(HWND_DESKTOP,hWnd,
          (PFNWP)DlgMsgProc,
          NULL,
          IDLG_MYDLG,
          (PSZ)CharStr);
```

In the dialog procedure, retrieve the string in the following manner:

```
strcpy(szLocalStr, (PSZ)mp2);
```

Attaching Data to a Window

Data is attached to a window by specifying a unique space or *data area* where the data can be placed. This enables a window to carry the data with it, which means the data is always available. Note, however, that the data area *must* be set aside when the window is registered. The data area is reserved by specifying its size in WinRegisterClass.

For example, to pass a pointer to a structure to a window, use a statement similar to the following:

```
rc = WinRegisterClass(hAB, (PCH)szClassName,
                      (PFNWP)WndProc,
                      CS_SIZEREDRAW,
                      sizeof(PVOID));          /*Space to reserve*/
```

Now, in the WM_CREATE statement or any other place in the window procedure where you want to manipulate the structure data, you can set and retrieve the structure pointer with WinSetWindowPtr and WinQueryWindowPtr. To use this method of attaching data, you must use the functions shown in Table 9-2.

Table 9-2. Functions That Attach Data

Function	Description
WinSetWindowPtr (hWnd, index, ptr);	Stores a pointer in the reserved window words. The *index* parameter is a zero-based index (LONG) that points to the stored pointers.
WinSetWindowText (hWnd,"A text string");	Sets the window text. If this function is called using a frame window handle, the text of the window's title bar is changed.
WinSetWindowULong (hWnd, index, value);	Stores a value (ULONG) in reserved memory at the position indicated by *index*.
WinSetWindowUShort(hWnd, index, value);	Stores a value (USHORT) in reserved memory at the position indicated by *index*.

To retrieve data in a window procedure, use the matching query functions (shown in Table 9-3). Because the window handle can be from a window in a different message queue, WinQueryWindowPtr and WinQueryWindow can be used to obtain data from windows in other threads.

Table 9-3. Query Functions

Function	Description
WinQueryWindowPtr (hWnd, index);	Retrieves a pointer stored in reserved memory at the position indicated by *index*.
WinQueryWindowText (hWnd, bufflen, buffer);	Retrieves the text into the buffer.
WinQueryWindowTextLength (hWnd);	Determines the buffer length required by WinQueryWindowText.
WinQueryWindowULong (hWnd, index);	Retrieves a pointer stored in reserved memory at the position indicated by *index*.
WinQueryWindowUShort	Retrieves a pointer stored in reserved memory at the position indicated by *index*.

Creating a Make File

All make files read a file containing all the modules and resources required to build the executable file and a list of dependencies. From this reading, the make file determines which files are out of date, automatically compiles these files, and relinks the system when all the modules are updated.

Many make files are given the same file name as the executable and an entension of *.MAK* (see Figure 9-4). To build these files, you would type **NMAKE filename.MAK** on the command line. To save keystrokes, rename your make file to MAKEFILE. This will enable you to compile your files by typing only the word **NMAKE**.

Figure 9-4. Submodule MAGICW.MAK

```
/*=========================== Magic Window Make File ==========================*/
CFLAGS  = /c /Se /Ss /Kc+ /Kp+ /Kb+ /Ti+ /DDEBUG
LFLAGS  = /CO /NOD /NOE /BASE:0x10000 /ALIGN:4
CC      = ICC
LIBS    = os2386.lib dde4mbs.lib
HELPIPF = magicw.ipf magicw1.ipf magicw.rc

# TARGET
magicw.exe : magicw.obj   magicwp1.obj magicdp1.obj magicdp2.obj syshelp.obj \
             msghndlr.obj os2f0004.obj os2f0006.obj os2f0007.obj os2f0008.obj \
             os2f0010.obj os2f011.obj  magicw.hlp   os2f0012.obj os2f0013.obj\
             os2f0014.obj os2f0015.obj magicw.res
   link386 @magicw.lnk;
   rc magicw.res magicw.exe

# RESOURCES
magicw.res : magicw.rc    menures.h   magicdlg.h   testdlg.h    helpids.h    \
             helpmenu.rc1 testdlg.dlg magicdlg.dlg magichlp.rc1 msghndlr.rc1 \
             help.dlg
   rc -r magicw.rc

# DEPENDENCIES
magicw.obj  : magicw.c
   $(CC) $(CFLAGS) magicw.c
magicwp1.obj : magicwp1.c menures.h magicdlg.h testdlg.h notebook.h listbox.h \
               dlgbox.h
   $(CC) $(CFLAGS) magicwp1.c
magicdp1.obj : magicdp1.c magicw.h magicdlg.h
   $(CC) $(CFLAGS) magicdp1.c
magicdp2.obj : magicdp2.c testdlg.h
   $(CC) $(CFLAGS) magicdp2.c
msghndlr.obj : msghndlr.c msghndlr.h
   $(CC) $(CFLAGS) msghndlr.c
syshelp.obj  : syshelp.c syshelp.h helpids.h
   $(CC) $(CFLAGS) syshelp.c
os2f0004.obj : os2f0004.c
   $(CC) $(CFLAGS) os2f0004.c
os2f0006.obj : os2f0006.c notebook.h
   $(CC) $(CFLAGS) os2f0006.c
os2f0007.obj : os2f0007.c notebook.h
   $(CC) $(CFLAGS) os2f0007.c
```

Submodule MAGICW.MAK (continued)

```
os2f0008.obj : os2f0008.c listbox.h
   $(CC) $(CFLAGS) os2f0008.c
os2f0010.obj : os2f0010.c
   $(CC) $(CFLAGS) os2f0010.c
os2f0011.obj : os2f0011.c
   $(CC) $(CFLAGS) os2f0011.c
os2f0012.obj : os2f0012.c
   $(CC) $(CFLAGS) os2f0012.c
os2f0013.obj : os2f0013.c
   $(CC) $(CFLAGS) os2f0013.c
os2f0014.obj : os2f0014.c
   $(CC) $(CFLAGS) os2f0014.c
os2f0015.obj : os2f0015.c
   $(CC) $(CFLAGS) os2f0015.c

# Build help (.IPF) file for Magic Window; compile to an intermediate file first

magicw.hlp : $(HELPIPF)
   icc -P magicw.ipf
   ipfc magicw.ipf
```

Note that in a make file the # character denotes a comment line. The parameters in capital letters at the head of the file (for example, CC and CFLAGS) are macros that expand to their full value when used. The use of macros ensures that it will be easy to modify a make file if something changes, such as a new compiler or library.

A make file compares the dates of objects on the left side of the colon (:) with the dates of objects on the right side of the colon to determine if a change has been made. It will recompile only the changed objects to create a new executable file. This technique is much faster than recompiling *all* of the objects.

TIP

To ensure that all changes are included, recompile your resources *every* time the executable file is rebuilt.

Once the recompile is complete, the executable (.EXE) file is built by linking all the object modules (in this case, the ones listed in the linker file in Figure 9-4). Application resources such as icons, pointers, and fonts should be recompiled if changed and then bound to the .EXE file.

If the make file has been correctly set up, the building of the .EXE file will be completely automated.

Additional Submodules for the Magic Window

The MAGIC submodules (such as the one in Figure 9-5) were written to be used with the Magic Window sample application, although they can be modified and used in other OS/2 applications (see Chapter 11, "The Warehouse Inventory Application"). Additional examples of submodules and header files are shown in Appendix A, "Reusable Submodules" and Appendix B, "Header Files". All of these modules are on the companion diskette.

Figure 9-5. Submodule MAGICWP1.C

```
/*====================== Magic Window Main Window Procedure ======================
FUNCTION:     MagicWndProc
DESCRIPTION:  Main window procedure - Processes all main window messages.
PARAMETERS:   Standard PM message processing
RETURNS:      MRESULT
EXPORTS:      MagicWndProc
================================================================================*/
#define STRINGLENGTH 50
#define INCL_PM

#include <os2.h>
#include <stdlib.h>
#include <string.h>

#include "magicdp.h"
#include "menures.h"
#include "magicdlg.h"
#include "testdlg.h"
#include "notebook.h"
#include "listbox.h"
#include "dlgbox.h"
#include "syshelp.h"
#include "magicw.h"
#include "helpids.h"
#include "magicwp1.h"
#include "msghndlr.h"
#include "os2f.h"

#define LISTENTRIES 26

FILEDLG StdFileDlg;

MRESULT EXPENTRY MagicWndProc(HWND hWnd, ULONG message, MPARAM mp1, MPARAM mp2)
{
   FONTMETRICS SystemFont;          /* System font metrics                      */
   HPS         hPS;                 /* Handle of presentation space to draw in */
   HWND        hFrameWnd;
   static ULONG ulResult;

   switch(message)
   {
      case WM_CREATE:
         hPS = WinGetPS(hWnd);
         GpiQueryFontMetrics(hPS, (LONG)sizeof(SystemFont), &SystemFont);
         return WMCreateMsg(hWnd);
```

Submodule MAGICWP1.C (continued)

```
      case WM_DESTROY:
         return WMDestroyMsg(hWnd);

      case WM_COMMAND:
         return WMCommandMsg(hWnd, message, mp1, mp2);

      case WM_CONTROL:
         return WMControlMsg(hWnd, message, mp1, mp2);

      case WM_CLOSE:
         hFrameWnd = WinQueryWindow(hWnd, QW_PARENT);
         if (hFrameWnd==hWndMain)
         {
            WinPostMsg(hWnd, WM_QUIT, 0L, 0L);
         }
         else
         {
            WinDestroyWindow(hFrameWnd);
         }
      break;

      case WM_PAINT:
         return (WMPaintMsg(hWnd));

      case WM_DRAWITEM:
         return WinDefWindowProc(hWnd, message, mp1, mp2);

      case WM_MEASUREITEM:
         return WinDefWindowProc(hWnd, message, mp1, mp2);

      default:
         /* Pass messages to help system */
         if (HlpCaseHM_Messages (message, mp1, &ulResult))
         {
            return (MPFROMSHORT(ulResult));
         }
         return WinDefWindowProc(hWnd, message, mp1, mp2);
   }
   return FALSE;
}
/*==============================WM_COMMAND MESSAGE ============================
FUNCTION:    WMCommandMsg
DESCRIPTION: Process any WM_COMMAND messages. The Presentation Manager messages
             for action bar and pull-down menu items are processed in this
             routine.
PARAMETERS:
RETURNS:     FALSE
============================================================================== */
MRESULT EXPENTRY WMCommandMsg(HWND hWnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
   USHORT         fUsrMsg = MB_OK | MB_MOVEABLE | MB_ICONEXCLAMATION;
   NOTEBOOK_DATA  Notebook;
   NOTEBOOK_DATA *NBPtr;
   PAGE_DATA      Page;
   PAGE_DATA     *PagePtr;
   USHORT         RtrnCode;
   HWND           hWndContainer;
   RECTL          Coords;
   CHAR           szAppNameC[] = {"S_CONTAINER"};
   CHAR           szTitleC[]   = {"Details Container Sample"};
   CHAR           szAppNameN[] = {"S_NOTEBOOK"};
```

Submodule MAGICWP1.C (continued)

```
CHAR           szTitleN[]   = {"Notebook Sample"};
CHAR           szAppNameL[] = {"S_LISTBOX"};
CHAR           szTitleL[]   = {"List Box Sample"};
HWND           hWndClient;
HWND           hWndFrame;
UCHAR          PathBuf[40];

HPS          hps;
ULONG        lTabHeight, lTabWidth;
FONTMETRICS  fm;

switch(SHORT1FROMMP(mp1))
{
   case IDM_ABOUT:
      WinDlgBox(HWND_DESKTOP,
                HWND_DESKTOP,
                AboutDlgProc,
                NULLHANDLE,
                IDD_ABOUT,
                NULL);
   break;

   case IDM_FILE_OPEN:
      memset(&StdFileDlg, 0, sizeof(StdFileDlg));
      StdFileDlg.cbSize   = sizeof(StdFileDlg);
      StdFileDlg.fl       = FDS_CENTER | FDS_OPEN_DIALOG;
      StdFileDlg.pszTitle = "Open a File";
      strcpy(StdFileDlg.szFullFile, "*.TXT");

      if (WinFileDlg(HWND_DESKTOP,
                     hWndMain,
                     &StdFileDlg)&& StdFileDlg.lReturn == DID_OK)
      {
         /* WinFileDlg returned valid path and filename  */
         /* Copy the filename into PathBuf               */

         strcpy(PathBuf,StdFileDlg.szFullFile);
         WinMessageBox (HWND_DESKTOP,
                        HWND_DESKTOP
                        &PathBuf,
                        "Retrieved File...",
                        OL, MB_OK);
      }
   break;

   case IDM_FILE_SAVE:
      memset(&StdFileDlg, 0, sizeof(StdFileDlg));
      StdFileDlg.cbSize   = sizeof(StdFileDlg);
      StdFileDlg.fl       = FDS_CENTER | FDS_OPEN_DIALOG;
      StdFileDlg.pszTitle = "Save a File";
      strcpy(StdFileDlg.szFullFile, "*.*");
      if (WinFileDlg(HWND_DESKTOP,
                     hWndMain,
                     &StdFileDlg)&& StdFileDlg.lReturn == DID_OK)
      {
         /* File dialog returned valid path and filename  */
         /* Copy the filename into PathBuf                */

         strcpy(PathBuf, StdFileDlg.szFullFile);
      }
   break;
```

Submodule MAGICWP1.C (continued)

```
   case IDM_OPTIONS_NOTEBOOK:                          /* Menu item "~Notebook" */
      WinRegisterClass (hAB, szAppNameN, (PFNWP)MagicWndProc, 0L, 0L);
      hWndFrame = CreateWindow((HWND)HWND_DESKTOP,
                               FCF_TITLEBAR | FCF_SYSMENU | FCF_SIZEBORDER,
                               szAppNameN,
                               szTitleN,
                               ID_FRAME_N,
                               100, 100, 350, 350,
                               &hWndClient,
                               0L,
                               SWP_SHOW);

   if (hWndFrame == (HWND)NULL)
   {
      DisplayMessage(IDS_USR_MSG2, HWND_DESKTOP, fUsrMsg ,NULL);
      break;
   }

   /* Establish addressability  */

   NBPtr                 = &Notebook;
   PagePtr               = &Page;
   NBPtr->ulNotebookStyle =
   BKS_SOLIDBIND      |       /* Solid binding                                  */
   BKS_BACKPAGESBR    |       /* Back pages intersect at bottom right corner */
   BKS_MAJORTABRIGHT  |       /* Major tabs positioned on right side          */
   BKS_SQUARETABS     |       /* Square tabs                                   */
   BKS_TABTEXTCENTER  |       /* Center tab text                               */
   BKS_STATUSTEXTLEFT;        /* Left-align, status-line text                  */

   NBPtr->NotebookID     = ID_BOOK;
   NBPtr->xLeft          = 20;
   NBPtr->xRight         = 300;
   NBPtr->yBottom        = 20;
   NBPtr->yTop           = 250;
   NBPtr->sxMajorTab     = 0;
   NBPtr->syMajorTab     = 0;
   NBPtr->sxMinorTab     = 0;
   NBPtr->syMinorTab     = 0;
   RtrnCode              = CreateWinNoteBook(hWndClient, NBPtr);

   /* Get tab sizing information from font metrics.  We doubled the font */
   /* sizes returned for the height and width values to give the tabs    */
   /* some white space.  If the tabs are going to be string values       */
   /* instead of a single letter, the width value should be calculated   */
   /* with a statement similar to the following:                         */
   /* lTabWidth = (fm.lAveCharWidth * 2) * strlen("Biggest Tab Title");  */

   hps = WinGetPS(NBPtr->hWndNotebook);
   GpiQueryFontMetrics(hps, (LONG) sizeof(fm), &fm);
   WinReleasePS(hps);
   lTabHeight = fm.lMaxBaselineExt * 2;
   lTabWidth  = fm.lAveCharWidth * 2;

   /* Set the size of the major tabs */

   WinSendMsg(NBPtr->hWndNotebook,
              BKM_SETDIMENSIONS,
              MPFROM2SHORT(lTabWidth, lTabHeight),
              MPFROMSHORT(BKA_MAJORTAB));
```

Submodule MAGICWP1.C (continued)

```
      if(RtrnCode)
      {
         DisplayMessage(IDS_USR_MSG2, HWND_DESKTOP, fUsrMsg, NULL);
         break;
      }
      NBPtr->NBPages[0] = (PAGE_DATA *)malloc(sizeof(PAGE_DATA));

      if(!NBPtr->NBPages[0])
      {
         DisplayMessage(IDS_USR_MSG3, HWND_DESKTOP, fUsrMsg, NULL);
         break;
      }
      NBPtr->NBPages[0] = (PAGE_DATA *)malloc(sizeof(PAGE_DATA));

      if(!NBPtr->NBPages[0])
      {
         DisplayMessage(IDS_USR_MSG3, HWND_DESKTOP, fUsrMsg, NULL);
         break;
      }
      PagePtr->usPageAttrib  = BKA_MAJOR | BKA_STATUSTEXTON | BKA_AUTOPAGESIZE;
      PagePtr->usPgPosn      = BKA_FIRST;
      PagePtr->StatusText    = "Page 1 of 2";
      PagePtr->TabText       = "A";
      NBPtr->NBPages[0]      = PagePtr;
      InsertNotebookPage(NBPtr, PagePtr);
      NBPtr->NBPages[1]      = (PAGE_DATA *)malloc(sizeof(PAGE_DATA));

      if(!NBPtr->NBPages[0])
      {
         DisplayMessage(IDS_USR_MSG3, HWND_DESKTOP, fUsrMsg, NULL);
         break;
      }
      PagePtr->usPageAttrib  = BKA_STATUSTEXTON | BKA_AUTOPAGESIZE;
      PagePtr->usPgPosn      = BKA_NEXT;
      PagePtr->StatusText    = "Page 2 of 2";
      PagePtr->TabText       = "";
      InsertNotebookPage(NBPtr, PagePtr);
      PagePtr->usPageAttrib  = BKA_MAJOR | BKA_AUTOPAGESIZE;
      PagePtr->usPgPosn      = BKA_NEXT;
      PagePtr->StatusText    = NULL;
      PagePtr->TabText       = "B";
      NBPtr->NBPages[1]      = PagePtr;
      InsertNotebookPage(NBPtr, PagePtr);
      break;

      case IDM_OPTIONS_DETAILS:
         WinRegisterClass (hAB, szAppNameC, (PFNWP)MagicWndProc, 0L, 0L);
         hWndFrame = CreateWindow((HWND)HWND_DESKTOP,
                                  FCF_TITLEBAR | FCF_SYSMENU | FCF_SIZEBORDER,
                                  szAppNameC,
                                  szTitleC,
                                  ID_FRAME_C,
                                  200, 200, 400, 250,
                                  &hWndClient,
                                  0L, SWP_SHOW);

         if (hWndFrame == (HWND)NULL)
         {
            DisplayMessage(IDS_USR_MSG2, HWND_DESKTOP, fUsrMsg, NULL);
            break;
         }
```

Submodule MAGICWP1.C (continued)

```
      Coords.xLeft      = 0;
      Coords.yBottom    = 0;
      Coords.xRight     = 392;
      Coords.yTop       = 250;

      hWndContainer     = CreateContainer(hWndClient, IDC_CONTAINER,
                                          (SHORT)Coords.xLeft + 200,
                                          Coords);

      if (hWndContainer != (HWND)NULL)
      {
         WinShowWindow(hWndContainer, TRUE);
      }
      else
      {
         DisplayMessage(IDS_USR_MSG2, HWND_DESKTOP, fUsrMsg, NULL);
      }
   break;

   case IDM_OPTIONS_MSGBOX:                      /* Menu item "~Message box" */
      WinMessageBox(HWND_DESKTOP, HWN_DESKTOP,
                    "This is an example of a message box.",
                    "Message Box Sample",
                    0L,                          /* ID is usually not used    */
                    MB_OK | MB_INFORMATION);
   break;

   case IDM_OPTIONS_LISTBOX:
      WinDlgBox (HWND_DESKTOP, hWndMain,
                 LBoxDlgProc,
                 NULLHANDLE,
                 IDD_LBOX, NULL);
   break;

   case IDM_OPTIONS_DLGBOX:
      WinDlgBox (HWND_DESKTOP, hWndMain,
                 MagicDlgMsgProc,
                 NULLHANDLE,
                 IDLG_DLGBOX, NULL);
   break;

   case IDM_OPTIONS_CONTROLS:
      WinDlgBox (HWND_DESKTOP, hWndMain,
                 CtrlDlgMsgProc,
                 NULLHANDLE,
                 IDLG_DISPLAY, NULL);
   break;

   case IDM_OPTIONS_EXIT_F3:                     /* Menu item "E~xit    \tF3" */
      if (WinMessageBox(HWND_DESKTOP, hWndMain,
          " Are you ready to exit the Magic Window Sample Program?",
          "Magic Window",
          0,
          MB_MOVEABLE | MB_ICONQUESTION | MB_YESNO | MB_DEFBUTTON1)==MBID_YES)
      {
         WinPostMsg(hWndMain,
                    WM_QUIT,
                    MPFROMLONG(NULL),
                    MPFROMLONG(NULL));
      }
   break;
```

Submodule MAGICWP1.C (continued)

```
      default:
         HlpCaseWM_Command(SHORT1FROMMP (mp1));        /* Service help messages */
         return WinDefWindowProc(hWnd, msg, mp1, mp2);
   }
   return(0L);
}

/*============================ WM_CREATE MESSAGE ================================
FUNCTION:    WMCreateMsg
DESCRIPTION: Process WM_CREATE messages. When window created, perform any
             initialization required.
PARAMETERS:  HWND - Handle of window.
RETURNS:     FALSE
============================================================================*/
MRESULT EXPENTRY WMCreateMsg(HWND hWnd)
{
   HPS hPS;                     /* Presentation space handle (space to draw in) */
   FONTMETRICS SystemFont;
   hPS = WinGetPS(hWnd);
   GpiQueryFontMetrics(hPS, (LONG)sizeof(SystemFont), &SystemFont);
   return (MRESULT)0;
}

/*============================ WM_DESTROY MESSAGE ===============================
FUNCTION:    WMDestroyMsg
DESCRIPTION: Process WM_DESTROY messages. When window destroyed, perform any
             cleanup required.
PARAMETERS:  Handle of window.
             Pointer to common storage area.
RETURNS:     FALSE
============================================================================*/
MRESULT EXPENTRY WMDestroyMsg(HWND hWnd)
{
   return 0L;
}

/*============================ WM_CONTROL MESSAGE ===============================
FUNCTION:    WMControlMsg
DESCRIPTION: Process WM_CONTROL messages.
PARAMETERS:  1.HWND - Client window handle  2.MPARAM - message mp1
RETURNS:     FALSE
============================================================================*/
MRESULT EXPENTRY WMControlMsg(HWND hWnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
   USHORT Result;
   if (HlpCaseHM_Messages((USHORT)(msg & 0xffff), mp1, &Result))
   {
      return ((MRESULT)Result);
   }
   return(WinDefWindowProc(hWnd, msg, mp1, mp2));
}

/*============================ WM_PAINT MESSAGE =================================
FUNCTION:    WMPaintMsg
DESCRIPTION: Process WM_PAINT messages.
RETURNS:     FALSE
============================================================================*/
MRESULT EXPENTRY WMPaintMsg(HWND hWnd)
{
   HPS    hPS;                  /* Handle for the presentation space        */
   RECTL  rClient;              /* Handle to rectangle formed by client area */
```

Submodule MAGICWP1.C (continued)

```
   hPS = WinBeginPaint(hWnd, 0L, 0L);
   WinQueryWindowRect(hWnd, &rClient);     /* Get and fill client area rectangle */
   WinFillRect(hPS, &rClient, CLR_BACKGROUND);
   WinEndPaint(hPS);
   return(0L);
}

/*============================= WM_DRAWITEM MESSAGE ==============================
FUNCTION:     WMDrawItemMsg
DESCRIPTION: Process WM_DRAWITEM messages when control is owner-drawn.
PARAMETERS:
RETURNS:      TRUE
================================================================================*/
MRESULT EXPENTRY WMDrawItemMsg(HWND hWnd,
                               FONTMETRICS  SystemFont,
                               ULONG        idControl,
                               MPARAM       mp2)
{
   INT         i;
   POINTL      pos;
   CHAR        szBuff[100];
   PSZ         token;
   static      int tabstop[] = {1,9};                          /* Entry spacing */
   POWNERITEM  pOwner;
   pOwner      = (POWNERITEM)mp2;

   if (((POWNERITEM) mp2)->fsState==((POWNERITEM) mp2)->fsStateOld)
   {
      /* Draw the item */
      i       = 0;
      pos.x   = ((POWNERITEM)mp2)->rclItem.xLeft;
      pos.y   = ((POWNERITEM)mp2)->rclItem.yBottom;
      pos.y  += SystemFont.lMaxDescender;

      /* Clear the line */
      WinFillRect(((POWNERITEM)mp2)->hps, &((POWNERITEM)mp2)->rclItem,
                  CLR_BACKGROUND);

      /* Retrieve the text */
      WinSendDlgItemMsg(pOwner->hwnd, idControl,
                        LM_QUERYITEMTEXT,
                        MPFROM2SHORT(pOwner->idItem, sizeof(szBuff)),
                        (MPARAM)szBuff);

      token = strtok(szBuff, "\t");
      while (token != NULL)
      {
         pos.x = pos.x + (tabstop[i++] * SystemFont.lAveCharWidth);
         GpiCharStringAt(((POWNERITEM)mp2)->hps, &pos, (LONG)strlen(token),
                         (PSZ) token);
         token = strtok(NULL, "\t");       /* Get next field */
      }
   }
   return((MRESULT)TRUE);                  /* Notify PM that drawing is complete */
}

MRESULT EXPENTRY AboutDlgProc (HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
   switch (msg)
   {
      case WM_COMMAND:
```

Submodule MAGICWP1.C (continued)

```
         switch (COMMANDMSG(&msg)->cmd)
         {
            case DID_OK:
               WinDismissDlg (hwnd,TRUE);
               return 0;
         }
      break;
   }
   return WinDefDlgProc (hwnd, msg, mp1, mp2);
}

BOOL FillBoxes(HWND hwnd)
{
   SHORT  rc;
   CHAR   szTemp[STRINGLENGTH];
   INT    i;

   for (i = IDS_LISTBOX1; i <= IDS_LISTBOX26;i++)
   {
      WinLoadString(OL, NULLHANDLE, i 20, szTemp);
      rc = (SHORT)WinSendMsg(hwnd,
                             LM_INSERTITEM,
                             MPFROMSHORT(LIT_SORTASCENDING),
                             MPFROMP((PSZ)szTemp));

      if (rc == LIT_ERROR || rc == LIT_MEMERROR)
      {
         DosBeep (100, 100);
         WinMessageBox(HWND_DESKTOP,
                       HWND_DESKTOP,
                       "List box error",
                       "Magic Window Sample PM Program",
                       0,
                       MB_MOVEABLE | MB_ICONHAND | MB_OK | MB_DEFBUTTON1);
         return FALSE;
      }
   }
   return TRUE;
}

MRESULT EXPENTRY LBoxDlgProc (HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
   switch (msg)
   {
      case WM_INITDLG:
         FillBoxes(WinWindowFromID(hwnd, IDC_LISTBOX1));
      break;

      case WM_COMMAND:
         switch (COMMANDMSG(&msg)-cmd)
         {
            case DID_OK:
               WinDismissDlg (hwnd, TRUE);
               return 0;
         }
      break;

   }
   return WinDefDlgProc (hwnd, msg, mp1, mp2);
}
```

Chapter 10.
Creating Effective Help for Your Application

Most graphical user interface (GUI) systems come with complete help programs, which enable you to give your users all the help information they need. These programs allow you to receive user feedback on their effectiveness and make changes quickly and easily to accommodate your user's needs.

Introduction to OS/2 Online Information

The OS/2 help and online programs are known as the *Information Presentation Facility (IPF)*. IPF is extremely powerful. It provides all the tools necessary to create effective help for your user and enables you to build tutorials and demonstrations that include text and animated graphics.

All IPF windows can be developed and viewed independently from any application in which they are used. This advantage enables you to obtain feedback from the end user ahead of the final system delivery, thereby creating a more efficient and personalized Help system. Because IPF allows immediate interaction between the user and the application, it might be helpful to have one person responsible for designing the IPF and for contacting the users to determine their help requirements. For example, one user might need only online help, while another might require a tutorial of the application or a reference to additional information such as a glossary of terms.

Online Information Features

Online information can be divided into two categories, authoring and programming, each of which uses different features. These features are provided with the OS/2 2.x Toolkit.

The *authoring* category uses the following features:

- A language that is capable of formatting text
- Hypertext links that allow nonlinear browsing of information
- A means of customizing windows
- An automatically generated index of the online information

- A compiler to create online documents and help windows
- A means of viewing the formatted online documents

In addition, OS/2 online help has two other features, *split windows* and *push buttons*. The text-formatting language, also called a *Tag language*, is used for push-button operation, window design and placement, and text and graphics display. Without using the Tag language, implementation of these facilities would require a lot of programming time and effort.

The *programming* category requires a method of displaying general help for your application windows and contextual help for fields within these windows, without having to code extensively.

The fundamentals of the OS/2 Tag language, which are necessary to display basic help, split windows, and create hypertext links, are described below. For a full discussion of the Tag language and its facilities, refer to the *Information Presentation Facility Guide and Reference* in hardcopy in the OS/2 2.0 Technical Library or online in the OS/2 2.x Toolkit.

Accessing the Online Tools

When you install the OS/2 2.x Toolkit, the CONFIG.SYS file is automatically updated. Therefore, it is important to create a backup of this file in case it is corrupted or destroyed. To do this, boot DOS and copy your CONFIG.SYS and any .INI files.

The following environment variables should exist in the CONFIG.SYS file to ensure access to the online tools:

BOOKSHELF=C:\OS2\BOOK	Specifies the path of the online documents. This variable is used by the VIEW command.
HELP=C:\OS2\HELP	Specifies the path of the help (.HLP) libraries.
IPFC=C:\TOOLKT20\IPFC	Specifies the path of the data files that are required by the IPF compiler.

NOTE

Do not attempt to compile and view online information if any of these environment variables are missing from the CONFIG.SYS file.

Using the IPF Compiler

Information is written using the Tag language and compiled using the IPF compiler. To start the compiler from the command line, use the following syntax:

```
IPFC filename[/INF][/S][/X][/Wn][> output filename]
```

where *filename* specifies the name and extension of your source file, */INF* compiles the source file as an online document, */S* suppresses the search function and increases the compression of compiled data by 10% to reduce storage requirements, */X* generates and displays a cross-reference list, */Wn* generates and displays a list of error messages of level *n*, and *output filename* specifies the file where the messages generated from /X and /W*n* are to be sent.

Once compiled, the information can be viewed independently of any system in which it is used. To do this, however, the text must be compiled as a document using the /INF switch, as shown in this example:

```
IPFC MYTEXT.IPF /INF
```

The above statement compiles the file MYTEXT and gives it an extension of *.INF*. The file then can be viewed by typing:

```
VIEW MYTEXT
```

Without the /INF switch, the compiler assumes it is compiling a help text window and gives the file an extension of *.HLP*, which cannot be viewed but can be used in the help library of your application.

Supporting International Languages

To create an international information or help display, use the following switches:

```
/COUNTRY  = nnn (country code)
/CODEPAGE = nnn (code page)
/LANGUAGE = xxx (international language file identifier)
```

The following example shows the syntax for compiling an online document for use with an international language (in this case, Spanish):

```
IPFC MYSTUFF /INF /COUNTRY=034 /CODEPAGE=437 /LANGUAGE=ESP
```

Concatenating Online Files

If an extremely large amount of information is to be viewed or if you wish to split your online documents into smaller units for maintenance purposes, you should concatenate your .INF files. Use the SET command to set an environment variable equal to the string of .INF file names you wish to concatenate, as follows:

```
SET MYDOCS=TEXT1.INF+TEXT2.INF+DIAGS.INF
```

Once this environment variable is set, the information within these files can be viewed by specifying the environment variable as the filename parameter of the VIEW command, as shown:

```
VIEW MYDOCS
```

Creating a Standard Window

Unless otherwise specified, the IPF compiler creates a standard window (see Figure 10-1) with the following components:

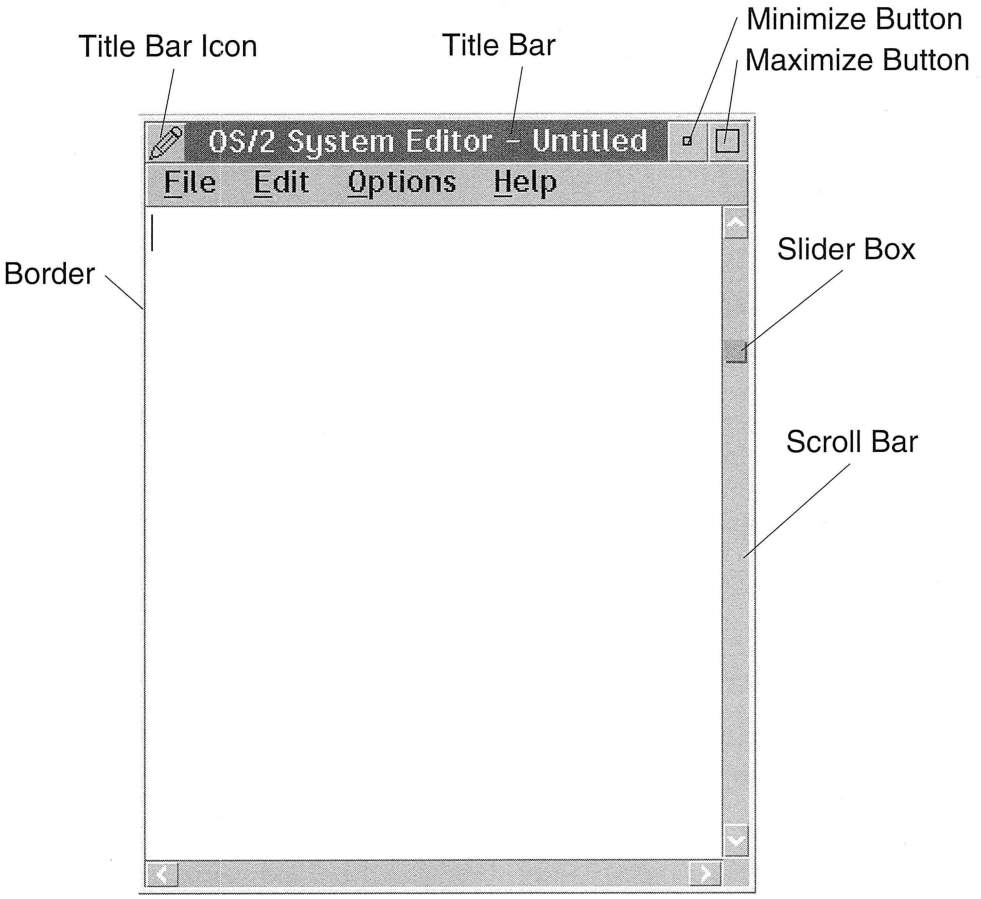
- Title bar
- Title bar icon
- Maximize and minimize buttons
- Menu bar
- Horizontal and vertical scroll bars
- Push buttons

The title bar icon and maximize and minimize buttons are used for sizing the window. The menu bar, scroll bars, and push buttons are used to manipulate the contents of the window.

The standard window can be customized to become the main help window (also known as a *coverpage* window) or a secondary help text window.

The main help window cannot be minimized; when opened, it is positioned so as to cover as little as possible of your application's window (see Figure 10-2 on page 182). Both the main help window and the help text window can be resized and moved. Text is automatically reformatted when a help text window is reduced or enlarged.

Figure 10-1. IPF Standard Window

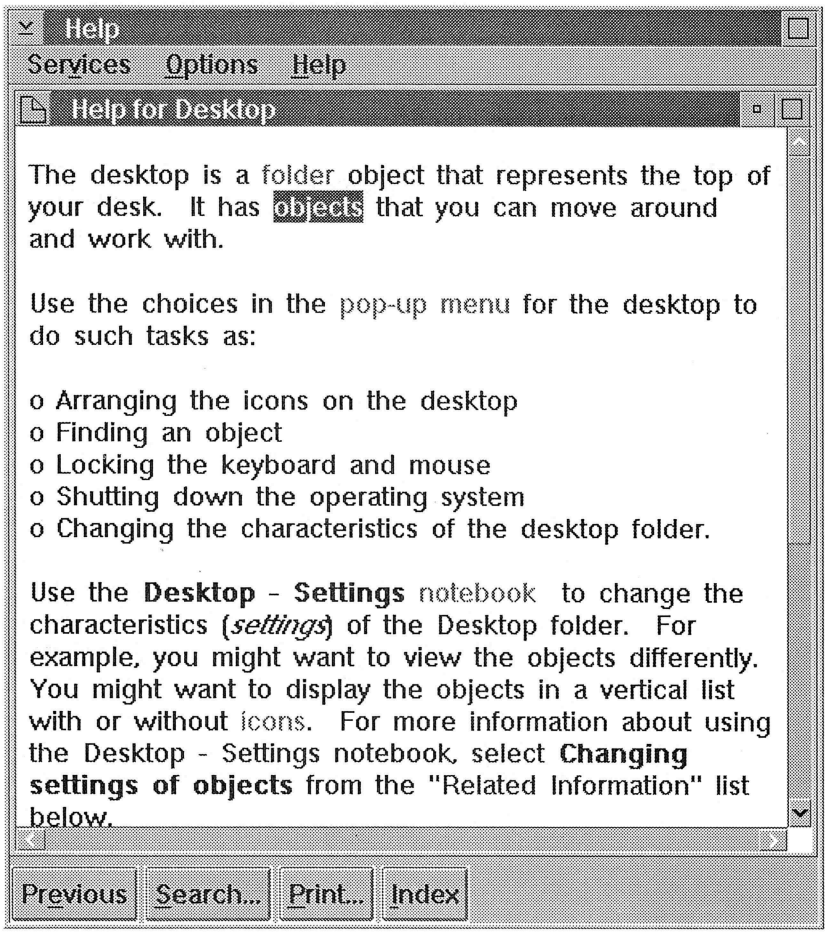


An online help menu (see Figure 10-3 on page 182) contains the following menu options:

- Help index
- General help
- Using help
- Keys help
- Product information
- Tutorial *

* A Tutorial option is available only if IPF is notified that an PM tutorial is included in your application.

Figure 10-2. Main Help Window



Keys help lists and describes any key combination assigned to an application function. It is up to you to provide for Keys help in your application. (Two examples of Keys help are provided on the companion diskette.)

Figure 10-3. Help Menu Options

Help index	F11
General help	F2
Using help	Shift + F10
Keys help	F9

Understanding the Tag Language Syntax

The IPF tagging language can customize windows, connect information units, and establish links to other applications. It also has some word processing capabilities.

IPF tags must start with a colon (:) and end with a period (.). Some tags, such as list or table tags, work in pairs and require a matching end tag. End tags always start with *:e* followed by the name of the start tag and a period.

Userdoc Tag

The tag *:userdoc.* is used to identify IPF source files for compiling purposes. This tag must be the first tag in the source file. All tagged text must reside between the :userdoc. start and end tags, as shown:

```
:userdoc.
start of tagged text
.
.
.
end of tagged text
:euserdoc.
```

The :userdoc. tag is placed in only one file for compiling. When compiling all of your IPF files, place the :userdoc. and :euserdoc. tags in a *Master* or *Base* file. Use the *.im* control word to imbed of all of your other IPF files in the Master file (see "Using Control Words and Special Characters" on page 187).

When compiling a single file for testing purposes, place the :userdoc. and :euserdoc. tags at the beginning and end of the file, respectively. Be sure to comment out these tags after testing.

WARNING

If the IPF compiler encounters a second :userdoc. tag, the compilation will stop.

Text String Tags

One of the most used tags is *:p.*, which formats a new paragraph. The :p. tag does not use an end tag. This tag and others, such as the heading tags, can have a string of characters associated with them.

A text string can be placed immediately following or under the tag to which it is associated, as shown in the following two examples of a heading tag:

```
:h1.Creating Effective Help for Your Application

:h1.
Creating Effective Help for Your Application
```

Using Tag Attributes

Most tags also can have attributes associated with them. For example, the following first level heading tag (*:h1.*) uses the attribute *res=*, which is a window identifier:

```
:h1 res=100.Further Examples
```

In the above example, *res=100* indicates the window resource to be used. The res= attribute can be a number from 1 to 64000. This number must be unique for each heading tag because there will be a corresponding define for it in a header file, such as the following:

```
#define HLP_XAMP1 100
```

Some attributes are designed to work with specific tags such as *:color fc=green*, which specifies the foreground color (*fc*) by using the value *green*. Other attributes have no values attached. For example, to create a *simple list* that has no blank lines between the list items (*:li.* tags), use the *:sl.* tag with the *compact* attribute (see Figure 10-4).

Figure 10-4. Simple List Tagging Example

```
:sl compact.
:li.Item 1
:li.Item 2
:li.Item 3
:li.Item 4
:esl.
```

Notice that the terminating period comes immediately after the attribute, not the tag, and that the :sl. tag requires a matching end (*:esl.*) tag.

Attributes can be specified on the same line as a tag or on lines that follow it. However, the entire attribute, including its value (for example, *break=all*), must be on one line.

To create an *ordered list*, use the *:ol.* and *:eol.* tags as shown in Figure 10-5.

Figure 10-5. Ordered List Tagging Example

```
:p.Assembly Instructions
:ol.
:li.For each drawer:
:ol compact.
:li.Screw metal dowels into holes in drawer front.
:li.Insert wooden dowels into remaining holes in drawer front.
:li.Fasten drawer back to drawer sides with screws.
:eol.
:li.Repeat procedure for remaining drawers.
:li.Insert drawers into chest frame.
:eol.
```

This tagging will display as shown in Figure 10-6. Notice that the IPF compiler inserts the correct numbers or letters for each list and automatically indents a *nested* list.

Figure 10-6. Ordered List Displayed

Assembly Instructions

1. For each drawer:

   a. Screw metal dowels into holes in drawer front.
   b. Insert wooden dowels into remaining holes in drawer front.
   c. Fasten drawer back to drawer sides with screws.

2. Repeat procedure for remaining drawers.

3. Insert drawers into chest frame.

A *definition list* contains terms and their descriptions. To create this type of list, use the *:dl.* and *:edl.* tags as shown in Figure 10-7. Notice that the *:li.* (list item) tag is replaced by the *:dt.* (definition term) and *:dd.* (definition description) tags. You can create headings for the terms and definitions by using the *:dthd.* (definition term heading) and *:ddhd.* (definition description heading) tags.

Figure 10-7. Definition List Tagging Example

```
:dl tsize=15 compact.
:dt.HM_ERROR
:dd.Example of a function name.
:dt.ULONG
:dd.Data type.
:dt.param1
:dd.Type of parameter.
:dt.flreply
:dd.Return code.
:edl.
```

Attributes also can be used with a definition list. The ones used in this example are *tsize=*, which controls the amount of space between the term and its definition, and *compact.*

This tagging example will display as shown in Figure 10-8.

Figure 10-8. Definition List Displayed

HM_ERROR	Example of a function name.
ULONG	Data type.
param1	Parameter type.
flreply	Return code.

Some attributes, such as *name=* or *text=*, can have values that contain spaces. In these cases, the value must be enclosed in single quotes, as shown in the following example:

```
:h1 id=400 res=400 name='My Application'.
```

Other tags cause specific boldface headings to be displayed, for example:

```
:nt.
This is a text string for a note.
:ent.
```

This note tag displays the word *Note:* in boldface type with the text in regular type, as shown:

Note: This is a text string for a note.

Two other boldface heading tags are warning and caution:

```
:warning.Warning text goes here.
:ewarning.

:caution.Discretionary text goes here.
:ecaution.
```

These tags display text as follows:

Warning: Warning text goes here.

CAUTION:
Discretionary text goes here.

Using Control Words and Special Characters

For special processing needs, the IPF compiler recognizes the following control words, which are placed at the beginning of a line:

.*	Comment text; do not interpret or display.
.br	Break (start a new line of) text.
.im filename	Imbed this file in the current file. The *.im* control word is used to print one large document from several smaller ones.

Imbedded files cannot contain the :userdoc. and :euserdoc. tags. These tags should only be used once in the Master file. Although any file can be imbedded into another file, we recommended that you imbed your files only into the Master file to ensure that all of them are accounted for and properly sequenced.

Special characters such as the colon (:), *at* sign (@), and number sign (#) are used in some of the IPF tags. To avoid confusion and possible error when these special characters are displayed as text, use only predefined symbols. These symbols usually start with an ampersand (&). For example, *&colon.*, *&atsign.*, and *&numsign.* cause the characters *:*, @, and # to be displayed, respectively.

TIP

You can create your own special symbols for words or phrases that you use repeatedly by using a macro called *NAMEIT* with its attributes *symbol=* and *text=*. An example of the format for this macro is as follows:

.nameit symbol=bktitle text='The OS/2 2.1 Corporate Programmer's Handbook'

When the file is compiled, everywhere the symbol *&bktitle.* is used in an .IPF file will be replaced by *The OS/2 2.1 Corporate Programmer's Handbook.*

A complete list of the IPF tags, control words, and special characters can be found in the *Information Presentation Facility Guide and Reference.*

Highlighting Online Text

Text can be highlighted (emphasized) by using nine tags, *:hp1.-:hp9.* with their corresponding end tags *:ehp1.- :ehp9.*.

For example:

```
This is a nonemphasized line.
:hp1.This is an italicized line.:ehp1.
```

These two lines display as follows:

This is a nonemphasized line.
This is an italicized line.

The tags :hp1. through :hp9. display as shown in Figure 10-9.

Figure 10-9. Tags Used for Highlighting

The :hp1. tag creates an italicized word or line.
The :hp2. tag creates a boldfaced word or line.
The :hp3. tag creates an italicized and boldfaced word or line.
The :hp4. tag creates a monospaced word or line.
The :hp5. tag creates an underscored word or line.
The :hp6. tag creates an italicized and underscored word or line.
The :hp7. tag creates a boldfaced and underscored word or line.
The :hp8. tag creates a monospaced word or line..
The :hp9. tag creates a monospaced word or line.

Tags :hp4., :hp8., and :hp9. can be highlighted with color as well as type style. To change the typeface of the text, use the *:font.* tag with the attributes *facename=* and *size=*.

Creating Window Titles

Window titles and headings are created using the *:title.* tag and heading tags (*:h1. - :h6*). Each :title. tag can have an accompanying text string, which becomes the name for the window. The text for the :title. tag must be kept to one line.

Each heading tag *must* have an associated text string and should be followed by a :p. tag (see Figure 10-10). The text strings of heading levels specified in the *toc=* attribute of the :docprof. tag (see "Determining the Contents Window Headings") will appear as headings and subheadings in the Contents window of an online document or help window.

Figure 10-10. Window Title Tagging Example

```
:userdoc.
:title.Your Online Document
:h1 id=550 res=550.Heading 1
:p.Text about this heading
:h2 id=650 res=650.Subheading 2
:p.Text about this subheading
:h3 id=750 res=750.Subheading 3
:p.Text about this subheading
:h2 id=850 res=850.Subheading 2
:p.Text about this subheading
:h3 id=950 res=950.Subheading 3
:p.Text about this subheading
:euserdoc.
```

Note that consecutive heading tags cannot have more than 16,000 characters between them or an error will occur.

To prevent a heading from appearing in the Contents window, use the *hide* attribute with any heading tag, as shown:

```
:h2 id=850 res=850 hide.Subheading 2
```

WARNING

Always display at least one heading. A compile error will occur if all the headings are hidden.

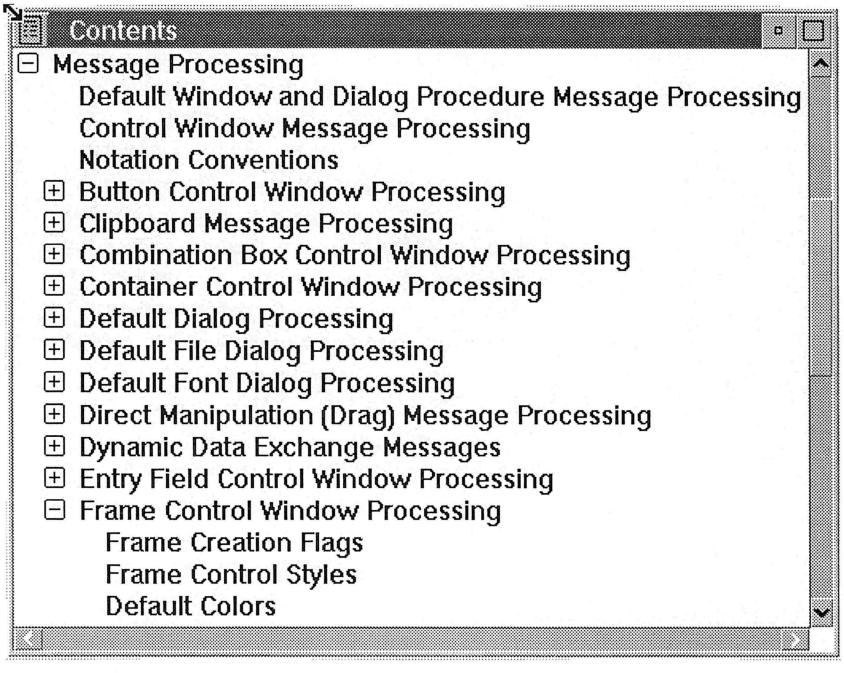
Determining the Contents Window Headings

By default, the heading tags that create entries in the Contents window and define the start of windows are *:h1.*, *:h2.*, and *:h3.* (see Figure 10-11 on page 190). This default can be changed by using the :docprof. tag with the toc= attribute. For example, specifying *:docprof toc=12.* tells the compiler that only heading levels 1 and 2 are to appear in the Contents window, whereas specifying *:docprof toc=1234.* tells the compiler to override the default and allow heading levels 1-4 to appear.

Creating Push Buttons

Push buttons are a feature of OS/2 2.1 provided by IPF to enable the user to accomplish commonly used tasks faster and more easily.

Figure 10-11. Contents Windows with Default Headings 1-3



The buttons are displayed in a *control area*, which can be defined within one of the two following windows:

- IPF coverpage window, which displays the help text for your application
- IPF text window, which is the secondary window of the coverpage window

The tagging example in Figure 10-12 shows how a simple coverpage help window can be set up.

Figure 10-12. Coverpage Help Window Tagging Example

```
:userdoc.
:title.Coverpage Help Window
:h1 id=1000 res=1000.Help Window
:p.This is text for online help.
:euserdoc.
```

Two sets of buttons are defined by IPF, one for online documents and the other for help windows (see Table 10-1).

Table 10-1. Defined Buttons

For Online Documents	For Help Windows
Back	
Contents	
Forward	
Index	Index
Previous	Previous
Print	Print
Search	Search
Tutorial (if used)	Tutorial (if used)

Although IPF displays the push buttons in the control area of a coverpage window by default, the push buttons in the control area of the help text window (see Figure 10-13) must be defined.

Figure 10-13. Push Buttons for Help Text Window



The *:ctrl.* tag determines which buttons to display and where to put them. When specifying a control area, always start with the :docprof. tag and imbed the :ctrl. tag between the *:ctrldef.* (control definition) start and end tags as shown:

```
:docprof toc=1234.
:ctrldef.
:ctrl.
:ectrldef.
```

To specify which push buttons to display, use the *controls=* attribute of the :ctrl. tag. The order that you specify the buttons will determine the order that the buttons are displayed.

The controls= attribute's values are shown in Table 10-2 on page 192. Notice that there is no selection for Tutorial because this button is displayed automatically if a tutorial is specified in the initialization structure *HMINIT*.

To display push buttons in the IPF coverpage window, use the *coverpage* attribute of the :ctrl. tag. To display push buttons in the text window, use the *page* attribute.

Table 10-2. Values for the Controls Attribute

Value	Description
BACK	Displays the previous page.
CONTENTS	Displays the Contents window.
ESC	Enables previously displayed information to be viewed. This value becomes the **Previous** push button.
FORWARD	Displays the next page.
INDEX	Displays the indexed topics.
PRINT	Displays a pop-up window for printing selected topics.
SEARCH	Displays a pop-up window for searching for a word or phrase.

To have a control area referenced by a heading tag, use the *ctrlid=* attribute. For example, a help window that displays the push buttons, **Previous** and **Forward**, in the control area of an IPF text window would use the tags shown in Figure 10-14.

Figure 10-14. Push Button Tagging Example

```
:docprof toc=123 ctrlarea=page.
:ctrldef.
:ctrl ctrlid=win6 controls='ESC FORWARD' page.
:ectrldef.
```

To override the default control area in a window, use the *ctrlarea=* attribute of the :docprof. tag.

The values for the ctrlarea= attribute are as follows:

both Specifies the control areas within the IPF coverpage window and the text window.

coverpage (Default) Indicates that the control area is at the bottom of the coverpage window.

none Indicates that push buttons are not to be used.

page Removes push buttons from the control area of the coverpage window.

Indexing with IPF Tags

Two tags are used by IPF for indexing: :i1. and :i2.. The *:i1.* tag describes the top-level or *primary* index entry. The *:i2.* tag describes the second-level or *secondary* index entry to the primary entry. The index entry text must be on the same line as the index tag. We recommend that at least one primary index entry be created for each window.

To create two levels of indexing, the id= attribute is used for the :i1. tag and the refid= attribute is used for the :i2. tag (see Figure 10-15).

Figure 10-15. Indexing Tagging Example

```
:i1 id=prog.programming in OS/2 2.1
:i2 refid=prog.traditional approach
:i2 refid=prog.modular approach
```

This tagging example will display as shown in Figure 10-16.

Figure 10-16. Indexing Displayed

```
programming in OS/2 2.1
   traditional approach
   modular approach
```

To ensure that subheadings are indexed correctly, the refid= attribute of the secondary index tag must have the same name as the id= attribute of the primary index tag to which it refers.

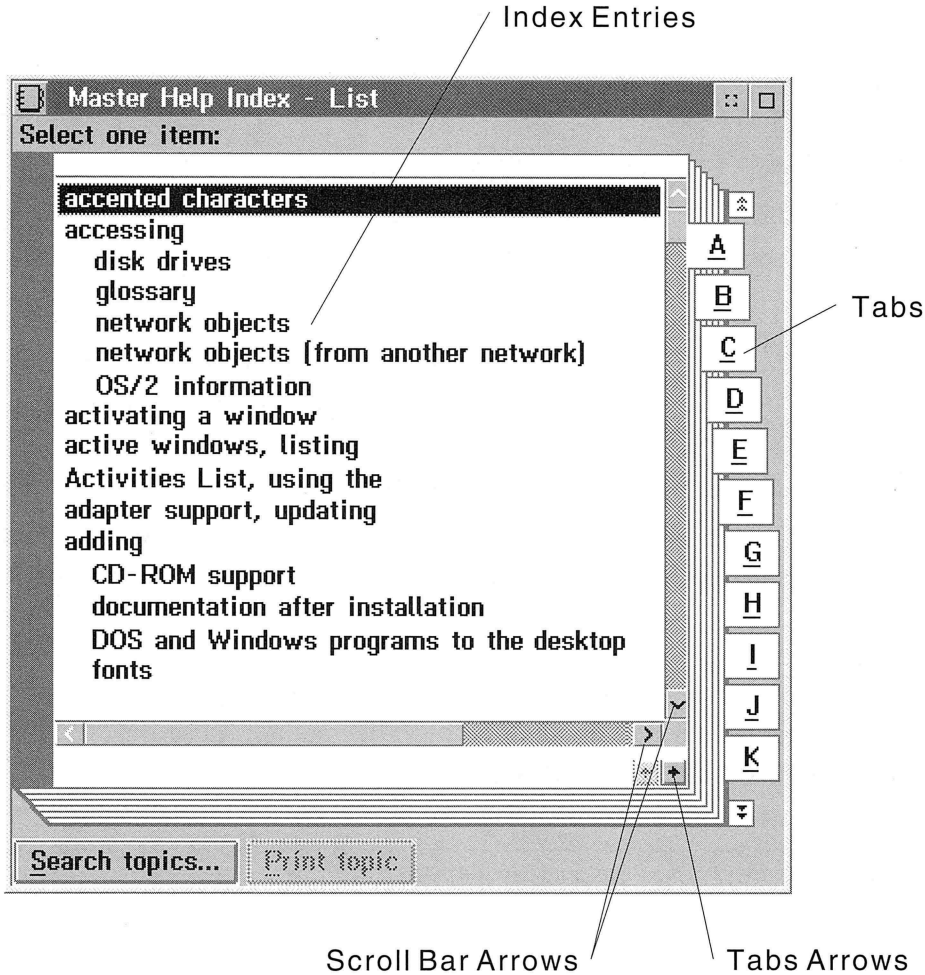
In the above example, the secondary index entries have the attribute *refid=prog*, which refers to the primary index entry's attribute *id=prog*. This method is used to create an index for both help windows and online documents.

Creating Entries for the Master Help Index

The *Master Help Index* is opened from the Desktop. It displays a notebook format that contains an alphabetized list of global index entries from the OS/2 help library. The Master Index enables you to quickly access help topics from multiple applications, thereby conserving system resources. New entries can be added to an existing Master Index or a new Master Index can be created.

The Master Help Index notebook has index tabs running down the right side. Selecting any one of these tabs will open a page that details the subject to which the tab refers (see Figure 10-17 on page 194).

Figure 10-17. Master Index Notebook



To create new entries for the Master Help Index, use the index tags :i1. and :i2. with the *global* attribute. Good choices for index entries would be major concepts and terms, common actions and procedures, restrictions or warnings, and acronyms.

TIP

When creating headings and indexes, it is considered good form to use the *gerund*, which is a noun derived from a verb and ends in *ing*. For example, use *password, using* instead of *password, use* and *creating a password* instead of *create a password*.

The tagging example in Figure 10-18 shows how these index entries can be set up.

Figure 10-18. Master Index Tagging Example

```
:i1 id=vmid global.accessing VM
:i2 refid=vmid global.using the Logon procedure
:i2 refid=vmid global.creating a password
```

To assist the user in searching for a topic, IPF provides for the use of synonyms. The synonym tag is *:isyn.*, which is used with the attribute *root=*.

Using the :Link. Tag

A *:link.* tag can display a footnote or help from a different help library (.HLP file), start another application, or send a message to the current application.

The :link. tag is used to navigate between source and target topics. This tag creates highlighted, selectable text or graphics that links the source topic to the target topic when the text or graphics is selected.

Creating Hypertext Links

Hypertext links enable *nonlinear* browsing of a document; that is, these links allow the user to switch to a different part of the document, perhaps to study a topic in greater depth, before continuing to browse. Clicking on a highlighted word or phrase causes a portion of the current document or a different document to be displayed.

A link from a word or phrase is set up as shown in Figure 10-19.

Figure 10-19. Linking from a Phrase

```
:link reftype=hd res=1000.
BACKUP FILES
:elink.
```

The corresponding link to the window is shown in Figure 10-20.

Figure 10-20. Linking to a Window

```
:h1 id=1000 res=1000 global.BACKUP FILES
:p.This window contains a list of your most recently backed up files.
```

In the above example, *reftype=hd* tells the compiler to link the phrase *BACKUP FILES* to a level-one heading in the target topic window that has a resource id (*res=*) of *1000*. The heading tag of the target window *must* contain the same res= identifier as the source window. When the user selects the highlighted phrase, the window defined by the resource id=1000 is displayed. Notice that the :link. tag has a matching end tag.

To link to another window in a target .INF or .HLP file in a different database, you must also use the *database=* attribute as shown in the example in Figure 10-21.

Figure 10-21. Linking to a Window in Another Database

```
:link reftype=hd database='os2/help/popup.hlp' res=1050.
Help Pop-Up Window
:elink.
```

The heading definition in the target file must have the *global* attribute as shown in Figure 10-22.

Figure 10-22. Target Heading with Global Attribute

```
:h1 id=1050 res=1050 global.
Help Pop-Up Window
```

If a link to a file is missing or not resolved, the word or phrase is deselected (not highlighted). However, it is dynamically highlighted if the link is added or resolved later.

To start another OS/2 application, use the :link. tag with the attributes *reftype=launch*, *object=*, and *data=*, as shown in Figure 10-23.

Figure 10-23. Starting Another OS/2 Application

```
:link reftype=launch object='D:\OS2\WIAPP.EXE' data=inv.
Warehouse Inventory Application
:elink.
```

The object= attribute indicates the path (file specification) of the application; the optional data= attribute specifies any parameters associated with the application.

To send a message to the current application, use the *inform=* attribute. This causes the *HM_INFORM* message to be sent. In this case, the res= attribute should refer to a resource ID for the application.

Creating Hypergraphic Links

Hypergraphic links link from a selectable graphic, usually a bit map, instead of from selectable text (see Figure 10-24).

Figure 10-24. Hypergraphics Tagging Example

```
:artwork name='mypic.bmp'.
:artlink.
:link reftype=hd res=1100.
:eartlink.
```

The *:artwork.* tag and its *name=* attribute identify the bit map. The *:artlink.* tag specifies the selectable areas of the bit map. Note that the *:eartlink.* tag replaces the need for an :elink. tag.

Creating Split Windows

It is often advantageous to display an object and have the explanatory text display next to it. IPF can do this without requiring extensive programming by using *automatic links*. These links occur when an action you selected uses a window in which a link is defined. The reftype= attribute of the :link. tag must have a value of *hd*, *launch*, or *inform*. The attributes *auto* and *split* also are required for automatic linking to secondary windows.

When a window is displayed, the links will automatically generate other windows. This effect is created by defining a window that contains the following:

- :h1. or :h2. primary-window heading tags without text. This is followed by automatic links to secondary windows.
- :h2. secondary window tags with text.

The size and position of the primary window determines the boundaries of the secondary windows. The sizes of secondary windows can be given as a percentage of the primary window. Note that if the size of a secondary window is absolutely defined and the values given exceed the size of the primary window, the secondary window will be truncated.

The primary window can contain only links to its secondary windows; it cannot contain text or graphics. Figure 10-25 is an example of the tags that are used for a primary window that contains a split window.

Figure 10-25. Split Window Tagging Example

```
:h1 res=1100 scroll=none.Primary Window
:link reftype=hd res=020 auto split group=5
      vpx=left vpy=top vpcx=50% vpcy=100%
      scroll=none titlebar=none.
:link reftype=hd res=022 auto split group=6
      vpx=right vpy=top vpcx=50% vpcy=100%
      scroll=vertical titlebar=none.
```

Because the primary window contains no text, it does not require a scroll bar and has a *scroll=none* attribute. Notice that the primary window in the above example has links to two secondary windows, each as high as the primary but half as wide. In these windows, the *scroll=* and *titlebar=* attributes define Presentation Manager controls. When these attributes are not defined, the default displays a window with all the trimmings (title bar, icon, min-max button, scroll bars, etc.).

The *group=* attribute is optional. If a group number is not assigned, IPF assigns the default *0*. This group number is used to determine if a window with the same group number is already open. If it is open, the text from the selected window is swapped to the opened window; otherwise, the selected window is opened. To always open a new window, use the *viewport* attribute.

Creating Footnotes

A *footnote* window is displayed when the user selects a hypertext word or phrase that is linked to that window by a *:fn.* tag. This tag has a matching end tag (*:efn.*). Any text between these two tags will appear in the footnote. The *id=* attribute is the footnote identifier.

A footnote pop-up window is created as shown in Figure 10-26.

Figure 10-26. Footnote Tagging Example

```
:fn id=modular.
Explanatory text goes here.
:efn.
```

Note that windows using the *split* attribute cannot contain footnotes, footnotes cannot contain index entries, and information contained in a footnote cannot be returned as the result of a search.

A hypertext link to a footnote is created as as shown in Figure 10-27.

Figure 10-27. Linking to a Footnote

```
:p.For more information about the modular approach, double-click on
:link refid=modular reftype=fn.
Modular Programming
:elink.
```

Adding Help to an Application

To add help facilities to an application, perform the following steps:

1. Set up the Help table and subtable; include the help constants in *PMHELP.H*. **Note:** Never alter the constants contained in this header file!
2. Initialize the *HELPINIT* structure (usually located in the main line of the program) during the IPF initialization phase.
3. Create a help instance.
4. Associate this help instance with the window chain of the application.
5. Respond to help messages generated by menu bar choices.
6. Destroy the help instance, when finished.

Setting Up the Help Tables

Two structures are required to identify window resources in the IPF library:

- The *Help table*, which associates each application window with its corresponding Help subtable and the window identifier (ID) of its extended help window.
- The *Help subtable*, which associates each window control, menu item, entry field, push button, etc. with the window identifier of its help window.

These structures can be defined either in memory or in resource (.RC) files. If the help tables are defined in memory, single entries can be changed dynamically. A new window ID that will be associated with a field or new fields that will be associated with existing windows can be added. If the help instance is in a resource file, it can be bound to the executable file or reside in a DLL. If it resides in a DLL, common help

is available to all running tasks. Note that the application must call DosLoadModule to load the DLL *before* creating the help instance.

The most popular method is to include the help in a resource file. To create a menu bar with one pull-down menu that offers two choices, you would use a resource file similar to the example in Figure 10-28. In this example, the Help table *MAIN_HELP-TABLE* associates the application windows (one of which happens to be an entry field) with their corresponding Help subtable and with the window identifier of its extended help window.

The Help subtables, *MAIN_WIN_HELP* and *NEW_KEYWORD_HELP*, associate the window control *ID_NEWKEYFLD* and the two menu items, *IDM_ADDPUB* and *IDM_ADDKEY*, with the ID of its associated help panel.

Figure 10-28. Example of a Resource File

```
#include <os2.h>
#include "examp01.h"

 MENU ID_MENU
 BEGIN
   SUBMENU "~Add", IDM_ADD
     BEGIN
       MENUITEM "Add ~Publication", IDM_ADDPUB
       MENUITEM "Add ~KeyWord", IDM_ADDKEY
     END
 END

 HELPSUBTABLE MAIN_WIN_HELP
 BEGIN
   HELPSUBITEM IDM_ADDPUB, ID_HELPPANEL1
   HELPSUBITEM IDM_ADDKEY, ID_HELPPANEL2
 END

 HELPSUBTABLE NEW_KEYWORD_HELP
 BEGIN
   HELPSUBITEM ID_NEWKEYWORD, ID_HELPPANEL3
 END

 HELPTABLE MAIN_HELPTABLE
 BEGIN
   HELPITEM ID_MENU, MAIN_WIN_HELP, ID_HELPPANEL4
   HELPITEM ID_NEWKEYFLD, NEW_KEYWORD_HELP, ID_HELPPANEL5
  END
```

For the entry field, the ID will either be sent as a parameter in a WinCreateWindow call or hard-coded into a dialog box template in the resource file. For example:

```
ENTRYFIELD  "", ID_NEWKEYWORD, 67, 36, 159, 8, ES_MARGIN
```

We recommend that conventional prefixes, such as *IDM* for menu identifiers, *ID* for resource IDs, etc., be used for the header file definitions.

The header file *EXAMP01.H* (see Figure 10-29) contains the ID definitions.

Figure 10-29. EXAMP01 Header File

```
#define BASE_ID                 0x1000
#define MAIN_HELPTABLE          BASE_ID + 2
#define MAIN_WIN_HELP           BASE_ID + 4
#define ID_MENU                 BASE_ID + 6
#define IDM_ADD                 BASE_ID + 8
#define IDM_ADDPUB              BASE_ID + 10
#define IDM_ADDKEY              BASE_ID + 12
#define ID_NEWKEYWORD           BASE_ID + 14
#define NEW_KEYWORD_HELP        BASE_ID + 16
#define ID_NEWKEYFLD            BASE_ID + 18
#define ID_HELPPANEL1           BASE_ID + 20
#define ID_HELPPANEL2           BASE_ID + 22
#define ID_HELPPANEL3           BASE_ID + 24
#define ID_HELPPANEL4           BASE_ID + 26
#define ID_HELPPANEL5           BASE_ID + 28
```

An example of the source file for a help windows is shown in Figure 10-30.

Figure 10-30. EXAMP01 Help Windows

```
:userdoc.
:h1 res=1200.This is Help Panel 1
:p.Display this help for IDM_ADDPUB.
:h1 res=1250.This is Help Panel 2
:p.Display this help for IDM_ADDKEY.
:h1 res=1300.This is Help Panel 3
:p.Display this help for the dialog box entry field.
:h1 res=1350.This is Help Panel 4
:p.Display this help for the extended help window.
:h1 res=1400.This is Help Panel 5
:p.Display this help for the extended help in the dialog box.
:euserdoc.
```

Creating the HELPINIT Structure

Unless the help is located in memory where it will be altered dynamically, the HELPINIT structure will usually be created in your application's main module during any initializing process. Before creating a help instance, HELPINIT must be initialized and allocated memory.

The HELPINIT structure can contain the parameters shown in Table 10-3 on page 202. (See the initialization function *OS2F0014.C* in Appendix A, "Reusable Submodules" for an example of how to initialize this structure.)

Table 10-3. HELPINIT Parameters

Data Type	Parameter	Description
ULONG	cb	Initialization structure length.
ULONG	ulReturnCode	Return code.
PSZ	pszTutorialName	Pointer to the tutorial name. If not NULL, IPF includes a tutorial choice in the Help pull-down.
PHELPTABLE	phtHelpTable	Pointer to the Help table.
HMODULE	hmodHelpTableModule	Name of the resource file that indexes the DLL containing the Help table. If the help is not in a DLL, this value is 0.
HMODULE	hmodAccelActionBarModule	Name of the DLL containing the modified menu bar. If not using a modified menu bar, this value is 0.
ULONG	idAccelTable	Name of the accelerator table if the modified menu bar is used. If not using a modified menu bar, this value is 0.
ULONG	idActionBar	Action bar template ID. If not using a modified menu bar, this value is 0.
PSZ	pszHelpWindowTitle	Pointer to window's title string, which can be changed dynamically by sending the message WM_SET_HELP_WINDOW_TITLE.
ULONG	fShowPanelId	Flag. If set to CMIC_SHOW_PANEL_ID, the window ID is displayed in the title bar of the help window. If set to CMIC_HIDE_PANEL_ID or 0, the window ID is not displayed. This parameter is useful for debugging.
PSZ	pszHelpLibraryName	Pointer to the name of the help library. Multiple names must be separated by a space.

Creating the Help Instance

To create the help instance, call the WinCreateHelpInstance function, which will pass the HELPINIT structure and return a handle to the help instance.

Associating the Help Instance

Once the help instance is created, the returned handle is passed as a parameter, along with the handle of the frame window, to WinAssociateHelpInstance to associate the help instance with the application window chain. Once the association has been made, help can be requested for any application window in the chain.

IPF normally communicates with the application window; the help window is positioned next to that window. The HM_SET_ACTIVE_WINDOW message enables the application to change the window with which IPF normally communicates. This ensures that the window to which HM_SET_ACTIVE_WINDOW is sent will receive any messages that are sent from IPF until the active window is changed by another HM_SET_ACTIVE_WINDOW message.

TIP

Some programmers have tried unsuccessfully to associate help with dialog boxes. However, an IPF instance can be associated only with application windows that have a *frame*. Because a dialog box is not a frame window, help instances cannot be associated with it. Therefore, when using dialog boxes, we recommend that you use the message HM_SET_ACTIVE_WINDOW.

Destroying the Help Instance

To end a help instance, call WinDestroyHelpInstance, which will pass the handle of the help instance to be destroyed (see Figure 10-31). This is usually done just before terminating the program.

Figure 10-31. Program Termination and Clean Up

```
WinDestroyWindow(hFrame);
WinDestroyMsgQueue(hmq);
WinTerminate(hab);
```

Most of the above code can generally be encapsulated into a single reusable function as shown in Figure 10-32.

Figure 10-32. Reusable InitHelp Function

```
/*============================== InitHelp ====================================
FUNCTION:     InitHelp
INPUTS:       1. Window handle
              2. Message parameter(mp2)
OUTPUTS:      Void
DESCRIPTION:  Initialize Help structure; associate help with primary frame window.
============================================================================*/
VOID InitHelp(HWND hWnd, MPARAM MsgParam)
{
   HELPINIT  HelpInit;                        /* Help initialization structure */
   HMODULE   hResId;
   HWND      hHelp;
   HWND      hFrame;
   HAB       hAB;
```

Reusable InitHelp Function (continued)

```
    hFrame   = WinQueryWindow(hWnd, QW_PARENT);
    hAB      = WinQueryAnchorBlock (hFrame);
    hResId   = (HMODULE)((PCREATESTRUCT)MsgParam)->pCtlData;

    HelpInit.cb                         = sizeof (HelpInit);
    HelpInit.ulReturnCode               = 0L;
    HelpInit.pszTutorialName            = "";
    HelpInit.phtHelpTable               = (PVOID)(0xFFFF0000 | MAIN_HELPTABLE);
    HelpInit.hmodHelpTableModule        = (HMODULE) hResId;
    HelpInit.hmodAccelActionBarModule   = (HMODULE) NULL;
    HelpInit.idAccelTable               = 0L;
    HelpInit.idActionBar                = 0L;
    HelpInit.pszHelpWindowTitle         = "Dialog Help";
    HelpInit.fShowPanelId               = CMIC_HIDE_PANEL_ID;
    HelpInit.pszHelpLibraryName         = "Dialog.Hlp";

    hHelp   = WinCreateHelpInstance (hAB,&HelpInit);
    WinAssociateHelpInstance(hHelp, hFrame);       /* Associate with frame window*/
}
```

Error Checklist

The lists that follow include some of the common errors that occur when creating help. Following each error is a checklist of reminders. All the suggestions have been explained earlier in this chapter.

Tagging Errors

- ☐ Is the period (.) placed after the tag's attribute?
- ☐ Are ids and refids spelled exactly the same?
- ☐ Are *res*= attributes associated correctly?
- ☐ Do all matching end tags begin with *:e*?
- ☐ Is a matching end tag missing?
- ☐ Are the *:docprof.* and *:edocprof.* tags used correctly?
- ☐ Are there more than 16,000 characters between any two consecutive heading tags?

Indexing Errors

- ☐ Do *:i2* tags follow the correct *:i1* tags?
- ☐ Are index ids and refids spelled exactly the same?

- ☐ Are topics and concepts correctly pointed to by index descriptions?
- ☐ Are *gerunds* used in index descriptions whenever possible?
- ☐ Have you tried to index a footnote?

Search Errors

- ☐ If synonyms are included in a search, is the *root=* attribute of the *:isyn.* tag correctly used?
- ☐ Are you trying to search for text contained in a footnote?

Hypertext Link Errors

- ☐ Is there a matching end tag (*:elink.*) for every *:link.* used?
- ☐ Are the *reftype=* attributes used correctly?
- ☐ When you are linking to another library, are the *database=* and *res=* attributes used correctly? Is the *global* attribute used with *id=* in the heading definition?
- ☐ When you are starting another application, does the *object* attribute indicate the right application? If used, does the *data* attribute specify the correct parameters?
- ☐ When you are sending a message to another application, does the *res=* attribute correctly refer to an application instead of a window?
- ☐ Have you tried to link a footnote?
- ☐ Has the *auto* attribute been specified for automatic linking to secondary windows?

Hypergraphic Link Errors

- ☐ Is there a matching end tag (*:eartlink.*) for every *:artlink.* used?
- ☐ Are the *reftype=* attributes used correctly?
- ☐ Is the name of the artwork, which follows immediately after the *name=* attribute, placed in single quotes?

Display Errors

- ☐ Is the resource id (*res*=) correct for the window to be displayed?
- ☐ Does text follow the automatic links to secondary windows?
- ☐ Is the size of a secondary window absolutely defined, and does it exceed the size of the primary window to which it is linked?
- ☐ Does the primary window contain text or graphics?
- ☐ Is the default displaying because the *scroll*= and *titlebar*= attributes are not correctly defined?
- ☐ Is the *group* attribute incorrectly assigned or assigned by default?

Footnote Errors

- ☐ Is there a matching end tag (*:efn.*) for every *:fn.* used?
- ☐ Have you ended one footnote before beginning another?
- ☐ Have you tried to index a footnote?
- ☐ Have you tried to link a footnote?

Miscellaneous Errors

- ☐ Have the Help table and subtable been set up? Is *PMHELP.H* included?
- ☐ Have the *constants* in the header files been altered in any way?
- ☐ Are conventional prefixes used for header file definitions? Has the *HELPINIT* structure been initialized? Does it contain the correct fields as depicted earlier in this chapter?
- ☐ Has a help instance been created and associated with the appropriate application window chain? Was *WinCreateHelpInstance* called correctly?
- ☐ Was the help instance destroyed before terminating the program?
- ☐ Is the application window associated with the correct window identifier?
- ☐ Are the window controls associated with the correct window identifier?

- ☐ If these structures are defined in a DLL, was *DosLoadModule* called before creating the help instance?
- ☐ If an ID was for an entry field, was it sent as a parameter in *WinCreateWindow* or *hardwired* into a dialog box template in the resource file?
- ☐ When IDs were numbered, was a *base ID* used? Was the base ID changed and not recompiled? Was the *res=* attribute in the help text changed at the same time?
- ☐ Was an attempt made to associate a help instance with a dialog box or an application window that does not have a frame? Was the *HM_SET_ACTIVE_WINDOW* message used as a solution to this problem?
- ☐ Was *error-checking* added to your functions as a safeguard?

Chapter 11.
The Warehouse Inventory Application

The Warehouse Inventory application is an example of how you can quickly build useable OS/2 2.1 applications by using the modules described in Chapter 9 and the submodules described in this chapter and in Appendix A, "Reusable Submodules".

To keep the Warehouse Inventory application within the scope of this book, we have used a linked list as a database with minimal error-checking. Although this would not be practical in the real world, it illustrates general design principles, including a few helpful tricks, and provides you with useable working code and a template for developing new systems. From this information, you should be able to see how to build more sophisticated systems using existing code as building blocks.

The Warehouse Inventory application also illustrates the most used, and most useful, controls and techniques. The database functions used in this application can easily be adapted to access a database manager, and the linked list is a useful structure in its own right. We have provided you a complete package, including memory initialization and management.

This sample was written as a single-threaded application. There are several places in the code that change the pointer to the "Wait" hourglass and back again. This practice should be avoided by starting a new thread, whenever practical. Although not generally used in simple applications such as this sample, the multi-threaded approach is the most desirable solution for complex applications.

You may wish to add some new functions to improve the usefulness of the Warehouse Inventory sample application. For example, the displayed inventory is a summary of a particular warehouse stock; updating is done item by item; and the linked list is kept current as items are added, deleted, or modified. However, to have a more in-depth, real-world application, you can add other functionality such as a list box pop up that displays a "pick list" to be printed and sent to the warehouse staff and drivers.

If you have insufficient memory problems, update the manifest constant LISTMAX (which is currently set to 4KB) in the database functions.

Warehouse Inventory Modules

The Warehouse Inventory application consists of the following module types:

- Procedure modules
- Function modules
- Function collection modules

Procedure Modules

winv010	Main	Main line
winv011	WINVDemoWndProc	Main window procedure
winv020	WhInvAddItem	Add item procedure
winv025	AboutDlgProc	About product
winv040	WhInvUpdateItem	Update item procedure
winv050	WhInvUtilIO	I/O module stub procedure

Function Modules

winv100	GetSelectedLine
winv110	GetSelectedItem
winv120	ClearWhseRecord
winv121	SetAddFieldLgths
winv122	GetAddDlgEntries
winv125	BuildWhseInvLine
winv126	AddInvItem
winv127	DeleteInventoryItem
winv128	UpdateInvItem
winv130	ListWhseInventory
winv132	ClearItemRecord
winv133	StopDB
winv135	StartDB
winv140	GetUpdDlgEntries
winv148	LoadWhseDBRecord
winv152	AddDBRec
winv153	UpdDBRec
winv154	DelDBRec

OS2f0001 RtAdjustCharString
OS2f0002 DisplayDBerror
OS2f0003 GetSelectedLBEntry
OS2f0004 CreateWindow
OS2f0026 UnsetSubMem

Function Collection Modules

msghndlr Message handler functions
lnklist Linked-list functions
syshelp Help system functions

Some of these modules are shown in the shaded Figure boxes in this chapter and in Appendix A, "Reusable Submodules." The header files for these modules are shown in Appendix B, "Header Files." All the modules and header files are found on the companion diskette.

Warehouse Inventory Main Module

The Warehouse Inventory application main module (see Figure 11-1 on page 212) is similar to the Magic Window main module, shown in Chapter 9. Both sample applications focuses on the Presentation Manager (PM) interface. Some applications spawn processes that require user interaction; each process might, therefore, have its own message queue. The sample illustrated here is a typical PM template and should suffice for the majority of applications that you intend to write.

A major difference between the Warehouse Inventory application's main module and the Magic Window main module is in how the window class is registered. In the Warehouse Inventory application, the last parameter is not NULL; instead, it requests private storage to be allocated for the window being registered. In the Warehouse Inventory main window procedure (WINVDemoWndProc), when the window is created using WMCreateMsg, the address of the warehouse record structure is set into this private storage by calling WinSetWindowPtr. When a control event takes place, this private storage is retrieved by calling WinQueryWindowPtr. In this sample application, only enough storage for a single pointer is requested; however, multiple pointers can be stored and retrieved in this manner, which is a better and safer solution to using global pointers.

Figure 11-1. Warehouse Inventory Main - WINV010.C

```
/*================================ WINV010.C ==================================
PROGRAM:    Warehouse Inventory Application This is the main module for the
            Warehouse Inventory application. It is started directly and creates
            the message-processing queue, binds to the database (and warns the
            user if this fails), registers the main window class, and then creates
            the window.

            When the window-processing procedure is entered, a list box is
            displayed that contains a summary of all the items in the Inventory
            database. The message-processing loop is then entered. When this
            returns False, the database is closed and the process returns to the
            application's main menu.

NOTE:       Throughout this code, reference is made to database access,
            however, this sample application uses a linked list to ensure that
            anyone can run it. It is assumed that most programmers will use IBM's
            Database Manager, but the Warehouse Inventory function modules can
            be adapted to any database environment.
PARAMETERS: Void
RETURNS:    INT
============================================================================*/
#define INCL_PM
#define INCL_WINHELP                        /* Help facility definitions        */
#define INCL_DOS

#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "dbdefs.h"                         /* DB field length definitions      */
#include "winvdefs.h"                       /* System definitions and constants */
#include "winvmsg.h"                        /* Message Ids                      */
#include "LLs_def.h"                        /* Linked-list definitions          */

#include "res010.h"                         /* Local resource ids               */
#include "res020.h"
#include "res040.h"
#include "msghndlr.h"                       /* Message facility header file     */
#include "glhelp.h"                         /* Help system variables            */
#include "winv010.h"                        /* Local variables and prototypes   */
#include "os2f.h"
#include "syshelp.h"

#include "win011.h"
#include "win130.h"
#include "win133.h"
#include "win135.h"

/* ============================ Data Declarations ============================*/
WHSE_RECORD  WhseRecord;
WHSE_PTR     WhsePtr;

WHSE_RECORD  xItemRecord;
WHSE_PTR     pWhse;

BOOL fDetail = FALSE                    /* Item detail display flag     */
CHAR szTitle[50];                       /* Window title                 */

DBASE WinvDB;
DBASE_MEM MemPtr;
```

Warehouse Inventory Main Module (continued)

```
HAB  hAB;                          /* Handle to anchor block                */
HMQ  hMQ;                          /* Handle to message queue               */
HWND hWndFrame;                    /* Handle to main window frame           */
HWND hWndClient;                   /* Handle to client window               */
HWND hWndInitMsg;                  /* Handle to initial message dialog box  */
HWND hWndListBox;                  /* Handle to Warehouse Inventory list box */
HWND hWndDetail;                   /* Handle to Item Detail dialog          */

/*====================== Warehouse Inventory Main Module ================== */
INT main(VOID)
{
   QMSG   qmsg;             /* MSG structure to store messages                  */
   PID    pid;              /* Process identifier for adding name to switch list */
   TID    tid;              /* Thread identifier                                */
   LONG   ret_code;         /* Error codes are negative numbers                 */
   ULONG  fCritMsg = MB_OK | MB_MOVEABLE | MB_CUACRITICAL;

   /* The WinInitialize routine initializes the PM facilities for use by this   */
   /* application and returns a handle to the anchor block assigned to the      */
   /* the application by Presentation Manager.                                  */

   WhsePtr = &WhseRecord;             /* Set warehouse inventory addressability */
   if((hAB = WinInitialize(0L)) == 0L)
   {
      return(FALSE);
   }

   /* The WinCreateMsgQueue call creates a message queue for this application. */

   if((hMQ = WinCreateMsgQueue(hAB, 0L)) == 0L)
   {
      return(FALSE);
   }

   /* The following function registers the class of an application window. */

   strcpy(szAppName, "WHINV010");
   if(!WinRegisterClass(hAB,
                        (PCH)szAppName,
                        (PFNWP)WINVDemoWndProc,
                        CS_SIZEREDRAW,
                        sizeof(WhsePtr)))
   {
      return(FALSE);
   }

   WinLoadString(hAB, 0L, IDS_WHSE_MENU_TITLE, sizeof(szTitle), szTitle);

   /* The CreateWindow function creates a frame window for this application's   */
   /* top window, and sets the window's size and location.                      */

   hWndFrame = CreateWindow((HWND)HWND_DESKTOP,
                            FCF_TITLEBAR   | FCF_SYSMENU    | FCF_MINBUTTON |
                            FCF_MAXBUTTON | FCF_SIZEBORDER | FCF_MENU      |
                            FCF_ICON       | FCF_ACCELTABLE | FCF_SHELLPOSITION,
                            szAppName,
                            szTitle,
                            ID_WINV010,
                            0, 1, 637, 250,
                            &hWndClient,
                            0L, SWP_SHOW);
```

Warehouse Inventory Main Module (continued)

```
   if(hWndFrame == 0L)
   {
      return(FALSE);
   }

   else
   {
      hWndUser = hWndFrame;                /* Store frame handle for help system */
   }

   HlpOpenHelp(hWndFrame, (ULONG)NULL);  /* Initialize help                      */
   WinSendMsg(hWndClient, WM_PAINT, 0, 0);

/*============================== USER-SPECIFIC ================================= */
   /* Bind to the database... */

   MemPtr   = &WinvDB;
   List     = &MemPtr->WhouseList;
   ret_code = StartDB(MemPtr);

   if (ret_code != OK)
   {
      if (ret_code == DB_NOTSTARTED)          /* Database not started      */
      {
         DisplayMessage(IDS_WHINV_MSG_32, HWND_DESKTOP, fCritMsg, NULL);
      }

      else
      {
         if (ret_code == DB_OVERLD)           /* Too many users             */
         {
            DisplayMessage(IDS_WHINV_MSG_37, HWND_DESKTOP, fCritMsg, NULL);
         }
         else
         {
            DisplayDBerror(ret_code);
         }
      }
   }

   if (ret_code != OK)
   (
       return(FALSE);
   )

/*==============================================================================*/

   /* Create list box invisible... */

   hWndListBox = WinCreateWindow(hWndClient,              /* Parent of window   */
                                 WC_LISTBOX,              /* Class of window    */
                                 "",                      /* Window text        */
                                 WS_SYNCPAINT |
                                 LS_OWNERDRAW,            /* Window style       */
                                 10, 10,                  /* x,y coordinates    */
                                 570, 197,                /* cx, cy coordinates */
                                 hWndClient,              /* Owner              */
                                 HWND_TOP,                /* Behind             */
                                 ID_WINVLIST,             /* List box window ID */
                                 NULL,                    /* Control data       */
                                 NULL);                   /* Pres. parameters   */
```

Warehouse Inventory Main Module (continued)

```
if (hWndListBox == 0L)                  /* Can't create window    */
{
   DisplayMessage(IDS_WHINV_MSG_02, HWND_DESKTOP, fCritMsg, NULL);
   return(FALSE);
}

/* Fill list box (time-consuming set up and DB reads), save current mouse */
/* pointer, and start new thread.                                          */

ret_code = ListWhseInventory(hWndListBox);

switch ((INT)ret_code)
{
   case OK:
   break;

   case TBL_EMPTY:
      DisplayMessage(IDS_WHINV_MSG_44, HWND_DESKTOP, fCritMsg, NULL);
      return(FALSE);

   case ERROR:
      DisplayMessage(IDS_WHINV_MSG_40, HWND_DESKTOP, fCritMsg, NULL);
      return(FALSE);

   default:
      DisplayDBerror(ret_code);
      return(FALSE);
}

/* Show the warehouse list */

WinShowWindow(hWndListBox, TRUE);
WinSetFocus(HWND_DESKTOP, hWndListBox);

/* The following in-line routine fills out the application's switch control  */
/* structure with the appropriate information to add the application's name  */
/* to the OS/2 Window List a list of the jobs currently running on the       */
/* computer.                                                                 */

WinQueryWindowProcess(hWndFrame,                    /* Frame window handle */
                      &pid,                         /* Process identifier  */
                      &tid);                        /* Thread identifier   */

pSwctl.hwnd          = hWndFrame;                   /* Frame window handle */
pSwctl.idProcess     = pid;                         /* Process identifier  */
pSwctl.uchVisibility = SWL_VISIBLE;                 /* Visibility          */
pSwctl.fbJump        = SWL_JUMPABLE;                /* Jump indicator      */

strcpy(pSwctl.szSwtitle,"Warehouse Inventory");     /* Frame window title  */

hSwitch = WinAddSwitchEntry(&pSwctl);

/* The following is the message loop for the application.                    */

while(WinGetMsg(hAB, (PQMSG)&qmsg, 0L, 0, 0))
{
   WinDispatchMsg(hAB,(PQMSG)&qmsg);
}

HlpDestroyHelp();                          /* Destroy current Help Instance */
```

Warehouse Inventory Main Module (continued)

```
    /* Perform clean up before exiting application. The following routine        */
    /* destroys the application's frame window (which also destroys its child   */
    /* windows), destroys its message queue, and disassociates the application  */
    /* from the Presentation Manager system.                                     */

    WinDestroyWindow(hWndFrame);     /* Destroy the frame window               */
    WinDestroyMsgQueue(hMQ);         /* Destroy this application's message clue */
    WinTerminate(hAB);               /* Terminate use of PM resources          */
    StopDB(MemPtr->BaseAddr);        /* Close the database                     */

    return (TRUE);
}   /* main() */
```

Warehouse Inventory Main Window Procedure

The general design of this procedure (see Figure 11-2) is similar to the main window procedure described in Chapter 9. Instead of a huge case statement, a small main module case statement is used for all the system messages that must be trapped. The application-specific code is placed in corresponding functions. This is an efficient solution that makes it easier for you to to copy and reuse functions.

The naming convention used here was chosen to avoid problems with the linker regarding duplicate names. If you wish to build even more generic systems, you can compile the message functions separately.

Passing Data to a Dialog Procedure

There are two areas of interest in the main window procedure: passing data to a dialog procedure and aligning columns of data in a list box. Data can be passed (or attached) to a window and retrieved during its creation by utilizing the private storage requested when the window was registered. In dialog procedures, the data is passed as a parameter of the WinDlgBox function.

For example, the WMCommandMsg function processes messages resulting from selections made from action bar and pull-down menus. In some of the message-processing code (for example, ADD_ITEM), you will see a reference to a structure pointer that contains the addresses of the warehouse and warehouse item records and the handle of the main list box. These addresses are required in some of the dialog procedures and are passed as the last parameter in WinDlgBox instead of NULL. This pointer is then retrieved from the second message parameter when the dialog is created.

Figure 11-2. Warehouse Inventory Main Window Procedure

```
/*================================ WINV0011.C ==================================
PROCEDURE:    WINVDemoWndProc
DESCRIPTION: This procedure provides service routines for the general PM events
             (messages) that Presentation Manager sends to the window, as well as
             the user-initiated events (messages) that are generated when the
             user selects the action bar and pull-down menu controls or the
             corresponding keyboard accelerators.

             The SWITCH statement distributes the window messages to the
             respective message service routines, which are serviced by the CASE
             statements.

             If a message is sent to this procedure that requires no action (no
             programmed CASE clause, i.e., no service routine), the message is
             defaulted to the WinDefWindowProc function, where it is disposed of
             by PM. It is essential to have the default processing handle all
             the messages that your procedure ignores.
PARAMETERS:  Standard PM message processing
RETURNS:     MRESULT
==============================================================================*/
#define INCL_PM
#define INCL_WINHELP                  /* Include IPF header file               */

#include <os2.h>
#include <stdlib.h>
#include <string.h>

#include "dbdefs.h"                   /* DB field length definitions           */
#include "winvdefs.h"                 /* System definitions and constants      */
#include "winvmsg.h"                  /* Message Ids                           */
#include "LLs_def.h"                  /* Linked-list definitions               */

#include "res010.h"                   /* Resource IDs                          */
#include "res020.h"
#include "res025.h"
#include "res040.h"

#include "winv.h"
#include "winv010.h"
#include "helpids.h"                  /* Help Ids                              */
#include "syshelp.h"                  /* Help system prototypes                */
#include "glhelp.h"                   /* Global help structure                 */
#include "os2f.h"
#include "msghndlr.h"

FONTMETRICS SystemFont;               /* System font metrics                   */
HPS  hPS;                             /* Handle of presentation space          */

/*============================================================================*/
MRESULT EXPENTRY WINVDemoWndProc(HWND hWnd, ULONG message, MPARAM mp1, MPARAM mp2)
{
   switch(message)
   {
      case WM_CREATE:
         return WMCreateMsg(hWnd);

      case WM_COMMAND:
         return WMCommandMsg(hWnd, message, mp1, mp2);

      case WM_CONTROL:
         return WMControlMsg(hWnd, message, mp1, mp2);
```

Warehouse Inventory Main Window Procedure (continued)

```
      case WM_CLOSE:
         if(hWnd != hWndClient)
         {
            break;
         }
         WinPostMsg(hWnd, WM_QUIT, 0L, 0L);
      break;

      case WM_PAINT:
         return WMPaintMsg(hWnd);

      case WM_MEASUREITEM:
         return WMMeasureItemMsg(hWnd);

      case WM_DRAWITEM:
         return WMDrawItemMsg(hWnd, SystemFont, mp2);

      default:
         /* Pass messages to help system */

         if (HlpCaseHM_Messages (message, mp1))
         {
            return(MPFROMSHORT(ulResult));
         }
         return WinDefWindowProc(hWnd, message, mp1, mp2);
   }
   return FALSE;
}

/*============================ WM_COMMAND Message ================================
FUNCTION:    WMCommandMsg
DESCRIPTION: Process WM_COMMAND messages. The PM messages for action bar and
             pull-down menu items are processed in this routine.
RETURNS:     FALSE
================================================================================*/
MRESULT EXPENTRY WMCommandMsg(HWND hWnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
   INT      RtrnCd;
   SHORT    Indx;                                    /* List-box line index    */
   USHORT   fErrMsg  = MB_OK       | MB_MOVEABLE | MB_ICONEXCLAMATION;
   USHORT   fQMsg    = MB_OKCANCEL | MB_MOVEABLE | MB_ICONQUESTION;
   CHAR     szBuff[BUFF_SIZE];
   SHORT    Response;
   PSZ      Token;
   static PARAMS PassedParams;

   switch(SHORT1FROMMP(mp1))

   {
      case IDM_OPTIONS_ADD_ITEM:                    /* Menu item "~Add item"   */
         hWndListBox           = WinWindowFromID(hWnd, ID_WINVLIST);
         PassedParams.hWndList = hWndListBox;
         PassedParams.pItemRec = &ItemRecord;
         WinDlgBox(HWND_DESKTOP, hWnd, (PFNWP)WhInvAddItem, 0L,
         IDLG_ADD_020,
                  &PassedParams);
      break;

      case IDM_OPTIONS_DELETE_ITEM:                 /* Menu item "~Delete item" */
         hWndListBox = WinWindowFromID(hWnd, ID_WINVLIST);
         Indx        = GetSelectedLine(hWndListBox);
```

Warehouse Inventory Main Window Procedure (continued)

```
      if (Indx == LIT_NONE)
      {
         /* Must select an item  */
         DisplayMessage(IDS_WHINV_MSG_41, HWND_DESKTOP, fErrMsg, NULL);
      }

      else
      {
         WinSendMsg(hWndListBox, LM_QUERYITEMTEXT,
                    MPFROM2SHORT(Indx, sizeof(szBuff)),
                    MPFROMP(szBuff));
         Token = strok(szBuff, "\t");

         if (Token !=NULL)
         {
            /* Do you want to delete it?  */
            Response = DisplayMessage(IDS_WHINV_MSG_29,
                                      HWND_DESKTOP,
                                      fQMsg,
                                      Token);
         }

         if (Response == MBID_OK)
         (
            /* Build search key           */

            Token = GetSelectedItem(hWndListBox, Indx);
            strcpy(WhseRecord.szWhouseKey, "01");
            strcpy(WhseRecord.szWhouseKey, Token);
            RtrnCd = DeleteInventoryItem(hWndListBox, Indx, &WhseRecord);

            if(RtrnCd == NOTFOUND)
               DisplayMessage(IDS_WHINV_MSG_39,
                              hWndListBox,
                              fErrMsg,
                              Token);
         )
      }
   break;

   case IDM_OPTION_EXIT_F3:         /* Menu item "E~xit  \tF3"    */
      WinPostMsg(hWnd, WM_QUIT, 0L, 0L);
   break;

   case IDM_OPTION_UPDT_ITEM:       /* Menu item "~Update item    */
      hWndListBox = WinWindowFromID(hWnd, ID_WINVLIST);
      Indx        = GetSelectedLine(hWndListBox);

      if(Indx == LIT_NONE)
      (
         /* Must select a warehouse item   */
         DisplayMessage(IDS_WHINV_MSG_41, hWndListBox, fErrMsg, NULL);
      )
      else
      {
         /* Build search key              */

         Token = GetSelectedItem(hWndListBox, Indx);
         strcpy(WhseRecord.szWhouseKey, "01");
         strcpy(WhseRecord.szWhouseKey, Token);
         RtrnCd = LoadWhseDBRecord(&WhseRecord);
```

Warehouse Inventory Main Window Procedure (continued)

```
            if(RtrnCd == NOTFOUND)
            (
               DisplayMessage(IDS_WHINV_MSG_39, hWndListBox, fErrMsg, Token);
            )
            else
            (
               PassedParams.pWhseRec = &WhseRecord;
               PassedParams.hWndList = hWndListBox;
               WinDlgBox(HWND_DESKTOP, hWnd,
                         (PFNWP)WhInvUpdateItem,
                         0L, IDLG_UPD_40,
                         &PassedParams);
            }
         }
      break;
      default:
         /* Service help messages */
         HlpCaseWM_Command(SHORT1FROMMP(mp1));
         return WinDefWindowProc(hWnd, msg, mp1, mp2);
   }
   return(0L);
}

/*============================== WM_CREATE Message ==============================
FUNCTION:    WMCreateMsg
DESCRIPTION: Process WM_CREATE messages. When window created, perform any
             initialization required.
PARAMETERS:  HWND - Handle of window.
RETURNS:     FALSE
================================================================================*/
MRESULT EXPENTRY WMCreateMsg(HWND hWnd)
{
   WhsePtr = &WhseRecord;
   WinSetWindowPtr(hWnd, 0, WhsePtr);
   hPS     = WinGetPS(hWnd);
   GpiQueryFontMetrics(hPS,(LONG)sizeof(SystemFont), &SystemFont);
   return (MRESULT) 0;
}

/*============================ WM_CONTROL Message ================================
FUNCTION:    WMControlMsg
DESCRIPTION: Process WM_CONTROL messages.
PARAMETERS:  1. HWND - Client window handle
             2. MPARAM - message mp1
RETURNS:     FALSE
================================================================================ */
MRESULT EXPENTRY WMControlMsg(HWND hWnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
   PSZ      Token;
   CHAR     szBuff[200];
   USHORT   Indx;
   WHSE_PTR pWhse;

   pWhseA  = WinQueryWindowPtr(hWnd, 0);

   if (SHORT1FROMMP(mp1) == ID_WINVLIST)
   {
      Indx = SHORT1FROMMR(WinSendDlgItemMsg(hWnd, ID_WINVLIST,
                          LM_QUERYSELECTION,
                          0L, 0L));
```

Warehouse Inventory Main Window Procedure (continued)

```
      WinSendDlgItemMsg(hWnd, ID_WINVLIST, LM_QUERYITEMTEXT,
                        MPFROM2SHORT(Indx, sizeof(szBuff)),szBuff);

      /* A selection has been made... Clear warehouse structure, store item    */
      /* number, and get new warehouse data.                                   */

      if (*szBuff[0] != 0x00)
      {
         ClearWhseRecord(pWhse);
         Token = strtok(szBuff, "\t");

         if (Token != NULL)
         {
            strcpy(pWhseA->szWhouseID, Token);
         }
      }
   }
   return ((MRESULT)HlpCaseHM_Messages(msg, mp1));            /* Process help */
}

/* ============================== WM_PAINT Message ==============================
FUNCTION:    WMPaintMsg
DESCRIPTION: Process WM_PAINT messages.
RETURNS:     FALSE
================================================================================*/
MRESULT EXPENTRY WMPaintMsg(HWND hWnd)
{
   POINTL pos;
   CHAR   szBuff[200];
   SHORT  width, ht;
   RECTL  rClient;

   hPS   = WinBeginPaint(hWnd, 0L, 0L);
   WinQueryWindowRect(hWnd, &rClient);
   WinFillRect(hPS, &rClient, CLR_BACKGROUND);

   /* Get position coordinates for start of column headings */
   pos.x = rClient.xLeft + 20L;
   pos.y = rClient.yTop - 27L;

   /* Get column headings string...  */
   WinLoadString(hAB, 0L, IDS_WHSE_MAIN_HDG, sizeof(szBuff), szBuff);

   /*  ...and write it in the client area  */
   GpiCharStringAt(hPS, &pos, LONG)strlen(szBuff), szBuff);

   WinEndPaint(hPS);

   /* Calculate new size and position of list box */
   width = (SHORT)(rClient.xRight - rClient.xLeft - 20L);
   ht    = (SHORT)(rClient.yTop - rClient.yBottom - 40L);

   WinSetWindowPos(hWndListBox,
                   HWND_TOP,
                   (SHORT)rClient.xLeft + 10,
                   (SHORT)rClient.yBottom + 10,
                   width, ht,
                   SWP_MOVE | SWP_SIZE);
   return(0L);
}
```

Warehouse Inventory Main Window Procedure (continued)

```
/*============================ WM_DRAWITEM Message ============================
FUNCTION:    WMDrawItemMsg
DESCRIPTION: Process WM_DRAWITEM messages. This message is trapped if the list
             box was created with the LS_OWNERDRAW style. The following code
             extracts each field (seperated by tab stops) from a string, and
             writes them in the list box in fixed-position columns.
RETURNS:     TRUE
============================================================================== */
MRESULT EXPENTRY WMDrawItemMsg(HWND hWnd, FONTMETRICS SystemFont, MPARAM mp2)
{
   POINTL pos;
   CHAR   szBuff[200];
   PSZ    Token;
   SHORT  i, j;
   static int tabstop[] = {1, 13, 48, 4, 13, 2, 10};

   if (((POWNERITEM) mp2)->fsState == ((POWNERITEM) mp2)->fsStateOld)
   {
      /* Draw the item...  */

      i      = 0;
      pos.x  = ((POWNERITEM)mp2)->rclItem.xLeft;
      pos.y  = ((POWNERITEM)mp2)->rclItem.yBottom;
      pos.y += systemfont.lMaxDescender;

      /* Clear the line...  */

      WinFillRect(((POWNERITEM)mp2)->hps,
                  &((POWNERITEM)mp2)->rclItem,
                  CLR_BACKGROUND);

      /* Retrieve the selected list-box line... */

      WinSendDlgItemMsg(hWnd,
                        ID_WINVLIST,
                        LM_QUERYITEMTEXT,
                        MPFROM2SHORT(((POWNERITEM)mp2)->idItem,
                        sizeof(szBuff)),
                        szBuff);

      Token = strtok(szBuff, "\t");           /* Get next field */
      j = 1;

      while (Token != NULL)
      {
         /* Set position of field and draw it... */

         if (j == 3 || j == 5)
         {
            /* Right align quantity and weight fields... */

            pos.x = pos.x + (tabstop[i++] * systemfont.lAveCharWidth);
            RtJustCharString(((POWNERITEM)mp2)->hps,
                             &pos,
                             (LONG)strlen(Token),
                             (PSZ)Token);
         }

         else
         {
            pos.x = pos.x + (tabstop[i++] * systemfont.lAveCharWidth);
```

Warehouse Inventory Main Window Procedure (continued)

```
            GpiCharStringAt(((POWNERITEM)mp2)->hps,
                            &pos,(LONG)strlen(Token),
                            (PSZ)Token);
         }

         Token = strtok(NULL,"\t");            /* Get next field */
         j++;
      }
   }
   return((MRESULT)TRUE);
   /* TRUE tells list box that you drew it to prevent PM from drawing it.  */
}

/*========================== WM_MEASUREITEM Message ============================
FUNCTION:     WMMeasureItemMsg
DESCRIPTION:  Process WM_MEASUREITEM messages. This message goes along with
              WM_DRAWITEM and is issued only once. It is required in order to
              obtain the maximum baseline extension so that the system can space
              text lines correctly.
RETURNS:      MRESULT - Maximum baseline extension.
==============================================================================*/
MRESULT EXPENTRY WMMeasureItemMsg(HWND hWnd)
{
   hPS = WinGetPS(hWnd);
   GpiQueryFontMetrics(hPS,
                       (LONG)sizeof(systemfont),
                       &systemfont);

   return((MRESULT)systemfont.lMaxBaselineExt);
}
```

Aligning Columns in a List Box

Columns can be aligned in a list box by using the *owner-draw* technique. Data displayed in a list box is usually thought of as columnar data. A list box knows how to draw itself, but does not have the ability to change text or colors. To use a different text font or color, you must create the list box with the owner-draw option and pass all drawing messages to its parent, which will handle all the drawing functions.

For example, the main list box is created with the WinCreateWindow function. Its class is WC_LISTBOX and its style is LS_OWNERDRAW, which is the parameter that tells the application that the owner is responsible for drawing the list box when it is required. If LS_OWNERDRAW is removed, your dialog window will not be notified to repaint.

To use the list box feature you must trap two messages: WM_DRAWITEM and WM_MEASUREITEM.

WM_DRAWITEM Message

The WM_DRAWITEM message is sent to the owner of a list box control each time an item is to be drawn. The WMDrawItemMsg function handles the actual drawing of text in the list box. Drawing and positioning of the column-heading text takes place during the processing of the WM_PAINT message, which prevents your headings from scrolling out of sight.

WMDrawItemMsg uses data extracted from the system structure *fState* to determine if a change has taken place and to obtain the new position of the presentation space. A call to WinSendDlgItemMsg, passing the identifier of the list box and the message LM_QUERYITEMTEXT, returns the selected list box text in a buffer.

The list box display lines are built in the BuildWhseInvLine function. This function concatenates the required data items into a string. Each item in the string is separated by a *tab* character in the application sample (any token can be used). The standard C-library function *strtok* is then used to break the string apart and the system GPI function *GpiCharStringAt* draws each item. The position is determined by the position obtained from the fState structure and the current tab-stop value multiplied by the average character width of the system font currently in use.

The initial adjusting of the tab-stop positions in the tab array can be done only by trial and error, which can be time-consuming. However, this function can be copied and used absolutely "as is" once you have tailored your display string to suit your own application requirements.

When the window containing the list box is resized, it is invalidated. This causes a WM_PAINT message to be sent to the client (parent). It is during the client window repainting that you must readjust the column-heading text. In the Warehouse Inventory sample application, the WMPaintMsg function takes care of this.

WM_MEASUREITEM Message

The message WM_MEASUREITEM returns the system font metrics. It is sent only once and is required. When the list box is created, this message is dispatched immediately. It enables the system to obtain the current font metrics, particularly the maximum base-line extension (descenders), by calling GpiQueryFontMetrics so that text can be properly spaced.

In the Warehouse Inventory application, text height is properly spaced by using the WMMeasureItemMsg function in the main window procedure, WinvDemoWndProc. The function RtJustCharString is used in aligning numeric fields. The actual client area is obtained by the WinQueryWindowRect function, which returns the new position in the RECTL structure. The text string is retrieved from the application resource file and is again written by the GPI function GpiCharStringAt.

The new position of the list box is also computed using the coordinates returned in the RECTL structure; WinSetWindowPos repositions it. This function can also be used virtually "as is", once the positional constants required for your own application have been determined (again by trial and error).

Note that the default case WinDefWindowProc in the main window procedure is essential; all procedures *must* have a default procedure in order to process messages that your application would not normally handle.

Dialog Procedures

Dialog procedures use a specific technique for retrieving data passed to the procedure during the call to WinDlgBox. For example, WMCreateMsg020 (see Figure 11-3). processes the WM_INITDLG message; part of this processing retrieves the passed data.

This process defines a pointer to a pointer and extracts the data that is obtained with the following:

```
ppItem =  PVOIDFROMMP(mp2);
pItem  = *ppItem;
```

The data is now available for any other function in the dialog procedure to access and use as needed.

Note that the default procedure WinDefDlgProc in the Warehouse Inventory Application's main module is absolutely required. WinDefDlgProc performs the same function as that of the default main window procedure.

Figure 11-3. Dialog Procedure WMCreateMsg020

```
/*============================== WINV020.C ====================================
FUNCTION:    WhInvAddItem
DESCRIPTION: Add inventory item message procedure. This procedure handles all the
             messages required when adding an inventory item to the database.
PARAMETERS:  Standard PM message processing
RETURNS:     MRESULT
EXPORTS:     AddInvMsgProc
============================================================================*/
#define INCL_PM
#define INCL_WINHELP                /* Include IPF header file          */

#include <os2.h>
#include <string.h>
#include <stdlib.h>
#include "dbdefs.h"                 /* DB field length definitions      */
#include "winvdefs.h"               /* System definitions and constants */
#include "winvmsg.h"                /* Message Ids                      */
#include "helpids.h"                /* Help facility Ids                */
#include "res020.h"                 /* Resource identifiers             */
#include "winv020.h"                /* Local variables and prototypes   */
#include "glhelp.h"                 /* Global help structure            */
#include "winv.h"
#include "msghndlr.h"
#include "os2f.h"
#include "syshelp.h"

/*============================================================================*/
MRESULT EXPENTRY WhInvAddItem(HWND hWndDlg, ULONG message, MPARAM mp1, MPARAM mp2)
{
   switch(message)
   {
      case WM_INITDLG :
         return WMCreateMsg020(hWndDlg, mp2);

      case WM_COMMAND :
         return WMCommandMsg020(hWndDlg, message, mp1 ,mp2);

      case WM_CONTROL:
         return WMControlMsg020(hWndDlg, message, mp1);

      case WM_CLOSE:
         WinDismissDlg(hWndDlg, message, mp1);
      break;

      default:
      /* Pass messages to help system */
      if (HlpCaseHM_Messages (message, mp1))
      {
         return(MPFROMSHORT(ulResult));
      }
      return(WinDefDlgProc(hWndDlg, message, mp1, mp2));
   }
   return FALSE;
}
/*============================== WM_CREATE Message ============================
FUNCTION:    WMCreateMsg020
DESCRIPTION: Process WM_INITDLG messages. When window created, perform any
             initialization required.
PARAMETERS:  HWND - Handle of window.  MPARAM - message parameter 2
RETURNS:     FALSE
============================================================================*/
```

Dialog Procedure (continued)

```
MRESULT EXPENTRY WMCreateMsg020(HWND hWndDlg, MPARAM mp2)
{
   ppItem = PVOIDFROMMP(mp2);                /* Get passed pointer */
   pItem  = *ppItem;
   ClearItemRecord(pItem.pItemRec);
   SetAddFieldLgths(hWndDlg);
   return(0L);
}

/*========================== WM_COMMAND Message ================================
FUNCTION:    WMCommandMsg020
DESCRIPTION: Process WM_COMMAND messages. The PM messages for action bar and
             pull-down menu items are processed in this routine.
PARAMETERS:  Standard message parameters
RETURNS:     FALSE
==============================================================================*/
MRESULT EXPENTRY WMCommandMsg020(HWND hWnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
   static HPOINTER hOldPtr, hWaitPtr;
   USHORT fCritMsg = MB_OK | MB_MOVEABLE | MB_CUACRITICAL;
   USHORT fErrMsg  = MB_OK | MB_MOVEABLE | MB_ICONEXCLAMATION;
   INT    Result;
   ULONG  RtrnCode = 0L;
   HWND   hndl;

   switch(SHORT1FROMMP(mp1))
   {
      case ADD_OK:
         /* Add inventory item data to database (time-consuming DB read), save  */
         /* current mouse pointer and change mouse pointer to "Wait" icon...    */

         hOldPtr  = WinQueryPointer(HWND_DESKTOP);
         hWaitPtr = WinQuerySysPointer(HWND_DESKTOP, SPTR_WAIT, FALSE);
         WinSetPointer(HWND_DESKTOP, hWaitPtr);
         Result   = AddInvItem(pItem.hWndList, pItem.pWhseRec);
         WinSetPointer(HWND_DESKTOP, hOldPtr); /* Restore original mouse pointer*/

            switch ((INT)Result)
            {
               case OK:
                  WinDismissDlg(hWnd,TRUE);
               break;

               case ERROR:
                  DisplayMessage(IDS_WHINV_MSG_20, hWnd, fCritMsg, NULL);
                  WinPostMsg(hWnd, WM_QUIT, 0L, 0L);
               break;

               case DUPLICATE:                /* Item already exists          */
                  DisplayMessage(IDS_WHINV_MSG_06, hWnd, fErrMsg, NULL);
                  hndl = WinWindowFromID(hWnd, IDLG_ADD_ITEM);
                  WinSetFocus(HWND_DESKTOP, hndl);
                  WinShowCursor(hndl, TRUE);
               break;

               case NO_ITEM_KEY:              /* No item number entered       */
                  DisplayMessage(IDS_WHINV_MSG_05, hWnd, fErrMsg, NULL);
                  hndl = WinWindowFromID(hWnd, IDLG_ADD_ITEM);
                  WinSetFocus(HWND_DESKTOP, hndl);
                  WinShowCursor(hndl, TRUE);
               break;
```

Dialog Procedure (continued)

```
                case NO_QTY:
                   DisplayMessage(IDS_WHINV_MSG_07, hWnd, fErrMsg, NULL);
                   hndl = WinWindowFromID(hWnd, IDLG_ADD_QTY);
                   WinSetFocus(HWND_DESKTOP, hndl);
                   WinShowCursor(hndl, TRUE);
                break;

                case NO_WGT:

                case NO_WGT_UNIT:
                   DisplayMessage(IDS_WHINV_MSG_08, hWnd, fCritMsg, NULL);

                   if(Result == (INT)NO_WGT)
                   {
                      hndl = WinWindowFromID(hWnd, IDLG_ADD_WT);
                      WinSetFocus(HWND_DESKTOP, hndl);
                   }

                   else
                   {
                      hndl = WinWindowFromID(hWnd, IDLG_ADD_WTUNIT);
                      WinSetFocus(HWND_DESKTOP, hndl);
                   }

                   WinShowCursor(hndl, TRUE);
                break;

                case NO_REORD:
                   DisplayMessage(IDS_WHINV_MSG_12, hWnd,
                                  fErrMsg, NULL);
                   hndl = WinWindowFromID(hWnd, IDLG_ADD_REORD);
                   WinSetFocus(HWND_DESKTOP, hndl);
                   WinShowCursor(hndl, TRUE);
                break;

                default:
                   DisplayDBerror(RtrnCode);       /* DB access error and quit   */
                   WinPostMsg(hWnd, WM_QUIT, 0L, 0L);
                break;
             }
          break;

       case ADD_CANCEL:
          /* Ignore data values entered and dismiss the dialog window */
          WinDismissDlg(hWnd, FALSE);
       break;

       case ADD_HELP:
          WinSendMsg(hWndHelpInstance,
                     HM_DISPLAY_HELP,
                     MPFROMSHORT(ADD_HELP),
                     MPFROMSHORT(HM_RESOURCEID));
       break;

       default:
          /* Service help messages */
          HlpCaseWM_Command(SHORT1FROMMP(mp1));
          return WinDefWindowProc(hWnd, msg, mp1, mp2);
    }
    return(0L);
}
```

Dialog Procedure (continued)

```
/*============================== WM_CONTROL Message ==============================
FUNCTION:     WMControlMsg020
DESCRIPTION:  Process WM_CONTROL messages.
PARAMETERS:   1.HWND - Client window handle
              2.MPARAM - message mp1
RETURNS:      FALSE
================================================================================*/
MRESULT EXPENTRY WMControlMsg020(HWND hWnd, ULONG msg, MPARAM mp1)
{
   GetAddDlgEntries(SHORT1FROMMP(mp1),
                    hWnd,
                    pItem.pWhseRec);
   return (MRESULT)HlpCaseHM_Messages(msg, mp1));              /* Process help */
}
```

Warehouse Inventory Submodules

The Warehouse Inventory application contains two types of submodules: generic and application-specific. The generic submodules are self-contained and can be used in most of the applications that you write for OS/2 2.1 with little or no changes made to them. (See Appendix A, "Reusable Submodules" and Appendix B, "Header Files").

The application-specific submodules have been written expressly for the Warehouse Inventory sample application. However, they can also be used for other applications with some modifications.

All the modules and submodules displayed in this chapter and elsewhere in this book can be found on the companion diskette.

Part IV. Reference Section

Chapter 12.
OS/2 Alphabetized Functions by Category

To save you the time and hassle of looking up the parameters of each function you might use in your application, we have provided you with a list of alphabetized OS/2 functions for the following categories:

- Dev functions (page233)
- Dos functions (page 234)
- Gpi functions (page 262)
- Prf functions (page306)
- Win functions (page308)

These functions have been updated for OS/2 2.1 and include their purpose, include file, function prototype, and parameters. For detailed information on other function categories and OS/2 functions, see the online information in the OS/2 2.1 Toolkit.

Dev Functions

DevCloseDC (Closes a device context)

```
#define INCL_DEV

HMF DevCloseDC(hdc);

HDC  hdc;     - Device-context handle
```

DevEscape (Allows escape calls to device drivers)

```
#define INCL_DEV

LONG DevEscape(hdc, lCode, lInCount, pbInData, plOutCount, pbOutData);

HDC     hdc;          - Device-context handle
LONG    lCode;        - Escape function to perform
LONG    lInCount;     - Input buffer count
PBYTE   pbInData;     - Pointer to input buffer
PLONG   plOutData;    - Pointer to output data count
PBYTE   pbOutData;    - Pointer to output data buffer
```

DevOpenDC (Opens a device context)

```
#define INCL_DEV

HDC DevOpenDC(hab, lType, pszToken, lCount, pdopData, hdcComp);

HAB           hab;          - Anchor-block handle
LONG          lType;        - Type of device context
PSZ           pszToken;     - Pointer to device-information token
LONG          lCount;       - Number of items
PDEVOPENDATA  pdopData;     - Pointer to open-device-context data area
HDC           hdcComp;      - Handle of compatible device context
```

DevPostDeviceModes (Returns and sets job properties)

```
#define INCL_WIN

LONG DevPostDeviceModes(hab, pdrivDriverData, pszDriverName, pszDeviceName, pszName,
                        flOptions);

HAB         hab;                - Anchor-block handle
PDRIVDATA   pdrivDriverData;    - Pointer to driver data
PSZ         pszDriverName;      - Pointer to string for device driver name
PSZ         pszDeviceName;      - Pointer to device type name
PSZ         pszName;            - Pointer to string for output device name
ULONG       flOptions;          - Specifies dialog options
```

DevQueryCaps (Retrieves device characteristics)

```
#define INCL_DEV

BOOL DevQueryCaps(hdc, lStart, lCount, alArray);

HDC    hdc;         - Device-context handle
LONG   lStart;      - First information item
LONG   lCount;      - Count of items
PLONG  alArray;     - Array for device capabilities
```

DevQueryDeviceNames (Retrieves device names and descriptions)

```
#define INCL_DEV

BOOL DevQueryDeviceNames(hab, pszDriverName, pldn, aDeviceName, aDeviceDesc, pldt,
                         aDataType);

HAB     hab;              - Anchor-block handle
PSZ     pszDriverName;    - Pointer to string for device driver name
PLONG   pldn;             - Max # of device names/descriptions that can be returned
PSTR32  aDeviceName;      - Array of device names
PSTR64  aDeviceDesc;      - Array of device descriptions
PLONG   pldt;             - Maximum number of data types that can be returned
PSTR16  aDataType;        - Pointer to array of data types
```

DevQueryHardcopyCaps (Retrieves hardcopy capabilities)

```
#define INCL_DEV

LONG DevQueryHardcopyCaps(hdc, lStartForm, lForms, phciHcInfo);

HDC      hdc;               - Device-context handle
LONG     lStartForm;        - Start forms code
LONG     lForms;            - Number of forms to query
PHCINFO  phciHcInfo;        - Pointer to hardcopy capabilities information
```

Dos Functions

DosAcknowledgeSignalException (Indicates a process wants to receive signals)

```
#define INCL_DOSEXCEPTIONS

APIRET DosAcknowlegeSignalException(SignalNumber);

ULONG   SignalNumber;     - Number of the signals to be acknowledged
```

DosAddMuxWaitSem (Adds a semaphore to a muxwait semaphore list)

```
#define INCL_DOSSEMAPHORES

APIRET DosAddMuxWaitSem(hmux, pSemRec);

HMUX        hmux;        - Handle of the muxwait semaphore
PSEMRECORD  pSemRec;     - Pointer to the SEMRECORD structure
```

DosAllocMem (Allocates a private memory object)

```
#define INCL_DOSMEMMGR

APIRET DosAllocMem(pBaseAddress, ObjectSize, AllocationFlags);

PPVOID  pBaseAddress;       - Pointer to memory object
ULONG   ObjectSize;         - Size of the memory object
ULONG   AllocationFlags;    - Memory object flags
```

DosAllocSharedMem (Allocates a shared memory object)

```
#define INCL_DOSMEMMGR

APIRET DosAllocSharedMem(pBaseAddress, pszName, ObjectSize, Flags);

PPVOID  pBaseAddress;     - Address of pointer to memory object
PSZ     pszName;          - Pointer to the name of the memory object
ULONG   ObjectSize;       - Size of the memory object
ULONG   Flags;            - Memory object flags
```

DosAsyncTimer (Creates an asynchronous timer)

```
#define INCL_DOSASYNCTIMER

APIRET DosAsyncTimer(TimerInterval, SemHandle, pHandle);

ULONG    TimeInterval;     - Time before semaphore is cleared
HSEM     SemHandle;        - Handle to event semaphore
PHTIMER  pHandle;          - Pointer to variable for timer handle
```

DosBeep (Generates a sound from the speaker)

```
#define INCL_DOSPROCESS

APIRET DosBeep(Frequency, Duration);

ULONG   Frequency;     - Frequency in hertz
ULONG   Duration;      - Duration in milliseconds
```

DosCallNPipe (Opens, reads from, writes to, and closes named pipe)

```
#define INCL_DOSNMPIPES

APIRET DosCallNPipe(pszFileName, pInBuffer, InBufferLen, pOutBuffer,
                    OutBufferLen, pBytesOut, Timeout);

PSZ     pszFileName;     - Pointer to pipe name
PVOID   pInBuffer;       - Pointer to input buffer
ULONG   InBufferLen;     - Number of bytes to be written
PVOID   pOutBuffer;      - Pointer to output buffer
ULONG   OutBufferLen;    - Maximum number of bytes in output buffer
PULONG  pBytesOut;       - Pointer to DWORD for number of bytes read
ULONG   Timeout;         - Timeout value
```

DosCancelLockRequest (Cancels DosSetFileLocks request)

```
#define INCL_DOSFILEMGR

APIRET DosCancelLockRequest(FileHandle, pLockRange);

HFILE      FileHandle;    - File handle
PFILELOCK  pLockRange;    - Pointer to structure describing region to be locked
```

DosClose (Closes a handle to a file, pipe, or device)

```
#define INCL_DOSFILEMGR

APIRET DosClose(FileHandle);

HFILE  FileHandle;    - File handle
```

DosCloseEventSem (Closes an event semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosCloseEventSem(hev);

HEV  hev;    - Handle of the semaphore
```

DosCloseMutexSem (Closes a mutex semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosCloseMutexSem(hmtx);

HMTX  hmtx;    - Handle of the mutex semaphore
```

DosCloseMuxWaitSem (Closes a muxwait semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosCloseMuxWaitSem(hmux);

HMUX  hmux;    - Handle of the muxwait semaphore
```

DosCloseQueue (Closes or deletes a queue)

```
#define INCL_DOSQUEUES

APIRET DosCloseQueue(QueueHandle);

HQUEUE  QueueHandle;     - Queue handle
```

DosCloseVDD (Closes virtual device driver handle)

```
#define INCL_DOSMVDM

APIRET DosCloseVDD(VDDHandle);

HVDD  VDDHandle;     - Virtual device driver handle
```

DosConnectNPipe (Waits for a client to open a pipe)

```
#define INCL_DOSNMPIPES

APIRET DosConnectNPipe(hplpeHandle);

HPIPE  hplpeHandle;     - Pipe handle
```

DosCopy (Copies a file or subdirectory)

```
#define INCL_DOSFILEMGR

APIRET DosCopy(pszSourceName, pszTargetName, OpMode);

PSZ    pszSourceName;     - Pointer to name of source file
PSZ    pszTargetName;     - Pointer to name of target file
ULONG  OpMode;            - Options
```

DosCreateDir (Creates a directory)

```
#define INCL_DOSFILEMGR

APIRET DosCreateDir(pszDirName, pEABuf);

PSZ     pszDirName;     - Pointer to directory name
PEAOP2  pEABuf;         - Pointer to structure for extended attributes
```

DosCreateEventSem (Creates an event semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosCreateEventSem(pszName, pphev, flAttr, fState);

PSZ    pszName;     - Pointer to ASCIIZ name of the semaphore
PHEV   pphev;       - Pointer to handle of the semaphore
ULONG  flAttr;      - Attributes of the semaphore
BOOL   fState;      - Initiate state of the semaphore
```

DosCreateMutexSem (Creates a mutex semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosCreateMutexSem(pszName, pphmtx, flAttr, fState);

PSZ    pszName;     - Pointer to ASCIIZ name of the semaphore
PHMTX  pphmtx;      - Pointer to handle of the semaphore
ULONG  flAttr;      - Attributes of the semaphore
BOOL   fState;      - Initiate state of the semaphore
```

DosCreateMuxWaitSem (Creates a muxwait semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosCreateMuxWaitSem(pszName, pphmux, cSemRec, ppSemRec, flAttr);

PSZ         pszName;      - Pointer to ASCII name of muxwait semaphore
PHMUX       pphmux;       - Pointer to handle of muxwait semaphore
ULONG       cSemRec;      - Count of the semaphore record entries
PSEMRECORD  ppSemRec;     - Pointer to the array of semaphore records
ULONG       flAttr;       - Attributes of the semaphore
```

DosCreateNPipe (Creates a named pipe and retrieves its handle)

```
#define INCL_DOSNMPIPES

APIRETDosCreateNPipe(pszFileName, pPipeHandle, OpenMode, PipeMode,
                     OutBufSize, InBufSize, Timeout);

PSZ     pszFileName;     - Pipe name
PHPIPE  pPipeHandle;     - Pointer to pipe handle
ULONG   OpenMode;        - Open mode of pipe
ULONG   PipeMode;        - Pipe-specific modes
ULONG   OutBufSize;      - Number of bytes in output buffer
ULONG   InBufSize;       - Number of bytes in input buffer
ULONG   Timeout;         - Timeout value
```

DosCreatePipe (Creates a pipe)

```
#define INCL_DOSQUEUES

APIRET DosCreatePipe(pReadHandle, pWriteHandle, PipeSize);

PHFILE  pReadHandle      - Pointer to variable for read handle
PHFILE  pWriteHandle     - Pointer to variable for write handle
ULONG   PipeSize         - Number of bytes reserved for pipe
```

DosCreateQueue (Creates and opens a queue)

```
#define INCL_DOSQUEUES

APIRET DosCreateQueue(pRWHandle, QueueFlags, pszQueueName);

PHQUEUE  pRWHandle;        - Pointer to variable for queue handle
ULONG    QueueFlags;       - Order in which elements are read-written
PSZ      pszQueueName;     - Pointer to queue name
```

DosCreateThread (Creates a thread)

```
#define INCL_DOSPROCESS

APIRET DosCreateThread(pThreadID, pThreadAddr, ThreadArg, ThreadFlags,
                       StackSize);

PTID       pThreadID;      - Pointer to variable for thread id
PFNTHREAD  pThreadAddr;    - Pointer to function
ULONG      ThreadArg;      - Parameter passed to thread
ULONG      ThreadFlags;    - Specifies when to start execution
ULONG      StackSize;      - Size of stack
```

DosDebug (Used to debug another application)

```
#define INCL_DOSPROCESS

APIRET DosDebug(pDbgBuf);

PVOID  pDbgBuf;    - Address of debugging buffer structure
```

DosDelete (Deletes a file name from a directory)

```
#define INCL_DOSFILEMGR

APIRET DosDelete(pszFileName);

PSZ  pszFileName;    - Pointer to string specifying path name
```

DosDeleteDir (Removes a directory)

```
#define INCL_DOSFILEMGR

APIRET DosDeleteDir(pszDirName);

PSZ  pszDirName;    - Pointer to directory name
```

DosDeleteMuxWaitSem (Deletes a semaphore from a muxwait semaphore list)

```
#define INCL_DOSSEMAPHORES

APIRET DosDeleteMuxWaitSem(hmux, SemHandle);

HMUX  hmux;          - Handle of the muxwait semaphore
HSEM  SemHandle;     - Handle of the semaphore to be deleted
```

DosDevConfig (Retrieves info about an attached device)

```
#define INCL_DOSPROCESS

APIRET DosDevConfig(pDeviceInfo, DeviceType);

PVOID  pDeviceInfo;    - Pointer to variable for device information
ULONG  DeviceType;     - Item number
```

DosDevIOCtl (Performs I/O control for devices)

```
#define INCL_DOSPROCESS

APIRET DosDevIOCtl(DevHandle, Category, Function, pParmlist, ParmLengthMax,
                   pParmLengthInOut, pDataArea, DataLengthMax, pDataLengthInOut);

HFILE   DevHandle;           - Device handle
ULONG   Category;            - Device category
ULONG   Function;            - Device-specific function code
PVOID   pParmList;           - Pointer to list of parameters
ULONG   ParmLengthMax;       - Maximum length of parameter list
PULONG  pParmLengthInOut;    - Pointer to length of parameters
PVOID   pDataArea;           - Pointer to data buffer
ULONG   DataLengthMax;       - Length of data buffer
PULONG  pDataLengthInOut;    - Pointer to length of data
```

DosDisConnectNPipe (Disconnects a named pipe)

```
#define INCL_DOS

APIRET DosDisConnectNPipe(hplpeHandle);

HPIPE  hplpeHandle;    - Pipe handle
```

DosDupHandle (Duplicates a handle)

```
#define INCL_DOSFILEMGR

APIRET DosDupHandle(OldFileHandle, pNewFileHandle);

HFILE   OldFileHandle;     - Handle of existing file
PHFILE  pNewFileHandle;    - Pointer to variable containing new file handle
```

DosEditName (Copies a string using editing semantics)

```
#define INCL_DOSFILEMGR

APIRET DosEditName(EditLevel, pszSourceString, pszEditString, pbTargetBufLen,
                   TargetBufLen);

ULONG  EditLevel;          - Edit level
PSZ    pszSourceString;    - Pointer to source string
PSZ    pszEditString;      - Pointer to editing string
PBYTE  pbTargetBufLen;     - Pointer to target buffer
ULONG  TargetBufLen;       - Length of target buffer
```

DosEnterCritSec (Suspends all but current thread)

```
#define INCL_DOSPROCESS

APIRET DosEnterCritSec(VOID);
```

DosEnterMustComplete (Enters code where asynchronous exceptions are held)

```
#define INCL_DOSEXCEPTIONS

APIRET DosEnterMustComplete(pNesting);

PULONG  pNesting;    - Pointer to value for current thread
```

DosEnumAttribute (Identifies extended attributes for a file)

```
#define INCL_DOSFILEMGR

APIRET DosEnumAttribute(RefType, pFileRef, EntryNum, pEnumBuf, EnumBufSize,
                        pEnumCnt, InfoLevel);

ULONG   RefType;        - Reference type
PVOID   pFileRef;       - Pointer to file name or handle
ULONG   EntryNum;       - Starting entry in list
PVOID   pEnumBuf;       - Pointer to data buffer
ULONG   EnumBufSize;    - Buffer size
PULONG  pEnumcnt;       - Pointer to number of entries to return
ULONG   InfoLevel;      - Info level
```

DosErrClass (Retrieves information about an error code)

```
#define INCL_DOSMISC

APIRET DosErrClass(Code, pClass, pAction, pLocus);

ULONG   Code;       - Error value for analysis
PULONG  pClass;     - Pointer to variable for error classification
PULONG  pAction;    - Pointer to variable for action
PULONG  pLocus;     - Pointer to variable for error origin
```

DosError (Enables/disables hard-error processing)

```
#define INCL_DOSMISC

APIRET DosError(Flags);

ULONG  Flags;    - Enable/disable error notification to users
```

DosExecPgm (Loads and starts a child process)

```
#define INCL_DOSPROCESS

APIRET DosExecPgm(pObjNameBuf, lObjNameBufL, ExecFlags, pszArgPointer,
                  pszEnvPointer, pReturnCodes, pszPgmPointer);

PCHAR         pObjNameBuf;      - Pointer to buffer for failed filename
LONG          lObjNameBufL;     - Length of failed filename buffer
ULONG         ExecFlags;        - Synchronous/trace flags
PSZ           pszArgPointer;    - Pointer to argument strings
PSZ           pszEnvPointer;    - Pointer to environment strings
PRESULTCODES  pReturnCodes;     - Pointer to structure for result codes
PSZ           pszPgmPointer;    - Pointer to file containing program to execute
```

DosExit (Terminates a thread or process)

```
#define INCL_DOSPROCESS

DosExit(ActionCode, ResultCode);

ULONG  ActionCode;    - Terminate current and/or all threads
ULONG  ResultCode;    - Result code for program
```

DosExitCritSec (Restores threads after calling DosEnterCritSec)

```
#define INCL_DOSPROCESS

APIRET DosExitCritSec(VOID);
```

DosExitList (Specifies functions to call upon termination)

```
#define INCL_DOSPROCESS

APIRET DosExitList(FunctionOrder, pRtnAddress);

ULONG        FunctionOrder;     - Function invocation order
PFNEXITLIST  pRtnAddress;       - Pointer to address of function
```

DosExitMustComplete (Exits code in which asynchronous exceptions are held)

```
#define INCL_DOSEXCEPTIONS

APIRET DosExitMustComplete(pNesting);

PULONG  pNesting;     - Pointer to value for current thread
```

DosFindClose (Closes a handle to a search directory)

```
#define INCL_DOSFILEMGR

APIRET DosFindClose(hdirDirHandle);

HDIR    hdirDirHandle;     - Handle of search directory
```

DosFindFirst (Finds first file objects for search)

```
#define INCL_DOSFILEMGR

APIRET DosFindFirst(pszFileName, pDirHandle, Attribute, pResultBuf,
                    ResultBufLen, pSearchCount, FileInfoLevel);

PSZ     pszFileName;       - Pointer to string specifying path name
PHDIR   pDirHandle;        - Pointer to handle
ULONG   Attribute;         - Search attribute
PVOID   pResultBuf;        - Pointer to structure receiving result
ULONG   ResultBufLen;      - Length of result buffer
PULONG  pSearchCount;      - Pointer to number of entries
ULONG   FileInfoLevel;     - Level of file information required
```

DosFindNext (Finds the next set of file objects in a search)

```
#define INCL_DOSFILEMGR

APIRET DosFindNext(hdirDirHandle, pResultBuf, ResultBufLen, pSearchCount);

HDIR    hdirDirHandle;     - Handle of search directory
PVOID   pResultBuf;        - Pointer to structure for search result
ULONG   ResultBufLen;      - Length of result buffer
PULONG  pSearchCount;      - Pointer to variable for file count
```

DosForceDelete (Removes a file name from a directory)

```
#define INCL_DOSFILEMGR

APIRET DosForceDelete(pszFileName);

PSZ  pszFileName;     - Pointer to the name of the file
```

DosFreeMem (Frees a memory object)

```
#define INCL_DOSMEMMGR

APIRET DosFreeMem(pBaseAddress);

PVOID  pBaseAddress;     - Pointer to the memory object to free
```

DosFreeModule (Frees a dynamic-link module)

```
#define INCL_DOSMODULEMGR

APIRET DosFreeModule(hmodModHandle);

HMODULE  hmodModHandle;     - Handle of dynamic link module to free
```

DosFreeResource (Frees a resource loaded by DosGetResource)

```
#define INCL_DOSMODULEMGR

APIRET DosFreeResource(pResAddr);

PVOID  pResAddr;     - Pointer to resource to free
```

DosFSAttach (Attaches or detaches a drive or device name)

```
#define INCL_DOSFILEMGR

APIRET DosFSAttach(pszDeviceName,pszFSDName,pDataBuffer,DataBufferLen,OpFlag);

PSZ     pszDeviceName;     - Pointer to device name
PSZ     pszFSDName;        - Pointer to file-system driver
PVOID   pDataBuffer;       - Pointer to buffer for file-system arguments
ULONG   DataBufferLen;     - Length of data buffer
ULONG   OpFlag;            - Operation type
```

DosFSCtl (Provides extended standard interface)

```
#define INCL_DOSFILEMGR

APIRET DosFSCtl(pDataArea, DataLengthMax, pDataLengthInOut, pParmList,
                ParmLengthMax, pParmLengthInOut, FunctionCode, pszRouteName,
                FileHandle, RouteMethod);

PVOID   pDataArea;           - Pointer to data area
ULONG   DataLengthMax;       - Buffer length
PULONG  pDataLengthInOut;    - Pointer to buffer for actual length
PVOID   pParmList;           - Pointer to parameter list
ULONG   ParmLengthMax;       - Maximum length of parameter list
PULONG  pParmLengthInOut;    - Pointer to buffer for actual length
ULONG   Functioncode;        - Function code
PSZ     pszRouteName;        - Pointer to file-system name
HFILE   FileHandle;          - File or device handle
ULONG   RouteMethod;         - Routing method
```

DosGetDateTime (Retrieves current date and time)

```
#define INCL_DOSDATETIME

APIRET DosGetDateTime(pPDateTime);

PDATETIME  pPDateTime;     - Pointer to structure for date and time
```

DosGetInfoBlocks (Returns the address of TIB and PIB)

```
#define INCL_DOSPROCESS

APIRET DosGetInfoBlocks(pptib, pppib);

PTIB  pptib;     - Address of pointer to a TIB
PPIB  pppib;     - Address of pointer to a PIB
```

DosGetMessage (Retrieves a message)

```
#define INCL_DOSMISC

APIRET DosGetMessage(plvTable, lvCount, pDataArea, DataLength, MsgNumber,
                     pszFileName, pMsgLength);

PCHAR   plvTable;        - Pointer to table of pointers to strings
ULONG   lvCount;         - Number of variable text strings
PCHAR   pDataArea;       - Pointer to buffer receiving message
ULONG   DataLength;      - Length of bytes in buffer
ULONG   MsgNumber;       - Message number to retrieve
PSZ     pszFileName;     - Pointer to name of file containing message
PULONG  pMsgLength;      - Pointer to length of bytes in returned message
```

DosGetNamedSharedMem (Obtains access to named shared memory)

```
#define INCL_DOSMEMMGR

APIRET DosGetNamedSharedMem(pBaseAddress, pszSharedMemName, AttributeFlags);

PPVOID  pBaseAddress;         - Address of pointer to memory object
PSZ     pszSharedMemName;     - Pointer to the name of the shared memory
ULONG   AttributeFlags;        - Attribute flags
```

DosGetResource (Retrieves address of resource object)

```
#define INCL_DOSMODULEMGR

APIRET DosGetResource(hmodModHandle, TypeID, NameID, pOffset);

HMODULE  hmodModHandle;     - Module handle
ULONG    TypeID;            - Resource type identifier
ULONG    NameID;            - Resource name identifier
PPVOID   pOffset;           - Pointer to variable for resource address
```

DosGetSharedMem (Obtains access to shared memory object)

```
#define INCL_DOSMEMMGR

APIRET DosGetSharedMem(pBaseAddress, AttributeFlags);

PVOID  pBaseAddress;       - Pointer to the memory object
ULONG  AttributeFlags;     - Attribute flags
```

DosGiveSharedMem (Gives memory access to other processes)

```
#define INCL_DOSMEMMGR

APIRET DosGiveSharedMem(pBaseAddress, pidProcessId, AttributeFlags);

PVOID  pBaseAddress;      - Pointer to base address of memory object
PID    pidProcessId;      - Process identifier
ULONG  AttributeFlags;    - Attribute flags
```

DosInsertMessage (Inserts text in a message)

```
#define INCL_DOSMISC

APIRET DosInsertMessage(plvTable, lvCount, pszMsgInput, sginLength, pDataArea,
                        DataLength, pMsgLength);

PCHAR   plvTable;       - Pointer to table of character pointers
ULONG   lvCount;        - Number of text strings
PSZ     pszMsgInput;    - Pointer to input message
ULONG   sginLength;     - Length of bytes in input message
PCHAR   pDataArea;      - Pointer to buffer for updated message
ULONG   DataLength;     - Length of bytes in buffer area
PULONG  pMsgLength;     - Pointer to variable for length of message
```

DosKillProcess (Terminates a process)

```
#define INCL_DOSPROCESS

APIRET DosKillProcess(ActionCode, pidProcessID);

ULONG  ActionCode;      - Flag for processes to be terminated
PID    pidProcessID;    - Pointer to process identifier of process to be ended
```

DosKillThread (Enables a thread to end another thread)

```
#define INCL_DOSPROCESS

APIRET DosKillThread(idThreadID);

TID  idThreadID;     - Identifier of thread to be ended
```

DosLoadModule (Loads a dynamic link module)

```
#define INCL_DOSMODULEMGR

APIRET DosLoadModule(pszObjNameBuf, ObjNameBufL, pszModName, pModHandle);

PSZ       pszObjNameBuf;     - Pointer to buffer for object name
ULONG     ObjNameBufL;       - Length of buffer for name
PSZ       pszModName;        - Pointer to module name
PHMODULE  pModHandle;        - Pointer to address of module handle
```

DosMapCase (Case maps characters in a string of binary values)

```
#define INCL_DOSNLS

APIRET DosMapCase(Length, pStructure, pBinaryString);

ULONG         Length;           - Length of string to case map
PCOUNTRYCODE  pStructure;       - Pointer to structure for country code
PCHAR         pBinaryString;    - Pointer to character string
```

DosMove (Moves a file object)

```
#define INCL_DOSFILEMGR

APIRET DosMove(pszOldPathName, pszNewPathName);

PSZ  pszOldPathName;     - Pointer to old path and filename
PSZ  pszNewPathName;     - Pointer to new path and filename
```

DosOpen (Opens or creates a file)

```
#define INCL_DOSFILEMGR

APIRET DosOpen(pszFileName, pFileHandle, pActionTaken, FileSize, FileAttribute,
               OpenFlag, OpenMode, pEABuf);

PSZ     pszFileName;       - Pointer to file name
PHFILE  pFileHandle;       - Pointer to variable for file handle
PULONG  pActionTaken;      - Pointer to variable for action taken
ULONG   FileSize;          - File size if created or truncated
ULONG   FileAttribute;     - File attribute
ULONG   OpenFlag;          - Action taken if file exists/does not exist
ULONG   OpenMode;          - Open mode of file
PEAOP2  pEABuf;            - Pointer to structure for extended attributes
```

DosOpenEventSem (Opens an event semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosOpenEventSem(pszName, pphev);

PSZ   pszName;     - Pointer to the ASCIIZ name of the semaphore
PHEV  pphev;       - Pointer to the event-semaphore handle to open
```

DosOpenMutexSem (Opens a mutex semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosOpenMutexSem(pszName, pphmtx);

PSZ    pszName;     - Pointer to the ASCIIZ name of the semaphore
PHMTX  pphmtx;      - Pointer to the mutex-semaphore handle to open
```

DosOpenMuxWaitSem (Opens a muxwait semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosOpenMuxWaitSem(pszName, pphmux);

PSZ    pszName;     - Pointer to the ASCIIZ name of the semaphore
PHMUX  pphmux;      - Pointer to the muxwait-semaphore handle to open
```

DosOpenQueue (Opens a queue)

```
#define INCL_DOSQUEUES

APIRET DosOpenQueue(pOwnerPID, pQueueHandle, pszQueueName);

PPID     pOwnerPID;        - Pointer to process ID for queue server process
PHQUEUE  pQueueHandle;     - Pointer to write handle of queue
PSZ      pszQueueName;     - Pointer to name of queue
```

DosOpenVDD (Opens a virtual device driver)

```
#define INCL_DOSMVDM

APIRET DosOpenVDD(pszVDDName, pVDDHandle);

PSZ    pszVDDName;     - Name of the virtual device driver
PHVDD  pVDDHandle;     - Pointer to the VDD handle
```

DosPeekNPipe (Reads a pipe without removing data)

```
#define INCL_DOSNMPIPES

APIRET DosPeekNPipe(hplpeHandle, pBuffer, BufferLen, pBytesRead, pBytesAvail,
                    pPipeState);

HPIPE       hplpeHandle;    - Pipe handle
PVOID       pBuffer;        - Pointer to output buffer for data
ULONG       BufferLen;      - Length of buffer for data
PULONG      pBytesRead;     - Pointer to number bytes read
PAVAILDATA  pBytesAvail;    - Pointer to number bytes available
PULONG      pPipeState;     - Pointer to variable for pipe state
```

DosPeekQueue (Retrieves a queue element without removing it)

```
#define INCL_DOSQUEUES

APIRET DosPeekQueue(QueueHandle, pRequest, pDataLength, pDataAddress,
                    pElementCode, NoWait, pbElemPriority, SemHandle);

HQUEUE        QueueHandle;      - Handle of queue to read from
PREQUESTDATA  pRequest;         - Pointer to structure for PID and request code
PULONG        pDataLength;      - Pointer to length of data
PPVOID        pDataAddress;     - Pointer to buffer for element received
PULONG        pElementCode;     - Pointer to variable for element position
BOOL          NoWait;           - Wait/no wait indicator
PBYTE         pbElemPriorty;    - Pointer to variable for priority of element
HEV           SemHandle;        - Semaphore handle
```

DosPhysicalDisk (Retrieves information about partitionable disks)

```
#define INCL_DOSPROCESS

APIRET DosPhysicalDisk(Function, pDataPtr, DataLen, pParmPtr, ParmLen);

ULONG  Function;    - Information to obtain
PVOID  pDataPtr;    - Pointer to output buffer
ULONG  DataLen;     - Output data buffer length
PVOID  pParmPtr;    - Pointer to input parameters
ULONG  ParmLen;     - Length of parameter buffer
```

DosPostEventSem (Posts an event semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosPostEventSem(hev);

HEV  hev;    - Semaphore handle
```

DosPurgeQueue (Purges a queue)

```
#define INCL_DOSQUEUES

APIRET DosPurgeQueue(QueueHandle);

HQUEUE  QueueHandle;     - Handle of queue to be purged
```

DosPutMessage (Writes a message)

```
#define INCL_DOSMISC

APIRET DosPutMessage(FileHandle, MessageLength, pMessageBuffer);

HFILE  FileHandle;        - Handle of output file/device
ULONG  MessageLength;     - Length of message buffer
PCHAR  pMessageBuffer;    - Pointer to message buffer
```

DosQueryAppType (Retrieves the type of an executable file)

```
#define INCL_DOSMODULEMGR

APIRET DosQueryAppType(pszExeFileName, pAppType);

PSZ     pszExeFileName;    - Pointer to executable-file name
PULONG  pAppType;          - Pointer to application-type flags
```

DosQueryCollate (Retrieves a collating sequence table)

```
#define INCL_DOSNLS

APIRET DosQueryCollate(Length, pctryc, pMemBuf, pDataLength);

ULONG         Length;         - Length of data area
PCOUNTRYCODE  pStructure;     - Pointer to structure containing country code
PCHAR         pMemBuf;        - Pointer to buffer for table
PULONG        pDataLength;    - Pointer to variable receiving table length
```

DosQueryCp (Retrieves a list of code pages for the process)

```
#define INCL_DOSNLS

APIRET DosQueryCp(Length, pCodePageList, pDataLength);

ULONG   Length;           - Length of bytes in buffer for list
PULONG  pCodePageList;    - Pointer to buffer receiving list
PULONG  pDataLength;      - Pointer to length of returned data
```

DosQueryCtryInfo (Retrieves country-dependent formatting information)

```
#define INCL_DOSNLS

APIRET DosQueryCtryInfo(Length, pStructure, pMemBuff, pDataLength);

ULONG         Length;         - Length of data area
PCOUNTRYCODE  pStructure;     - Pointer to structure containing country info
PCOUNTRYINFO  pMemBuff;       - Pointer to structure receiving country info
PULONG        pDataLength;    - Pointer to length of returned data
```

DosQueryCurrentDir (Retrieves the path of the current directory)

```
#define INCL_DOSFILEMGR

APIRET DosQueryCurrentDir(DriveNumber, pbDirPath, pDirPathLen);

ULONG   DriveNumber;     - Drive number
PBYTE   pbDirPath;       - Pointer to buffer receiving directory path
PULONG  pDirPathLen;     - Pointer to variable receiving length of path
```

DosQueryCurrentDisk (Retrieves letter of the current default drive)

```
#define INCL_DOSFILEMGR

APIRET DosQueryCurrentDisk(pDriveNumber, pLogicalDriveMap);

PULONG  pDriveNumber;         - Pointer to default drive number
PULONG  pLogicalDriveMap;     - Pointer to variable receiving drive map
```

DosQueryDBCSEnv (Retrieves the DBCS environment vector)

```
#define INCL_DOSNLS

APIRET DosQueryDBCSEnv(Length, pStructure, pMemoryBuffer);

ULONG         Length;           - Length of buffer
PCOUNTRYCODE  pStructure;       - Pointer to structure for country code
PCHAR         pMemoryBuffer;    - Pointer to buffer for DBCS information
```

DosQueryEventSem (Retrieves the number of postings to a semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosQueryEventSem(hev, pPostCt);

HEV     hev;         - Semaphore handle
PULONG  pPostCt;     - Pointer to count of current posts
```

DosQueryFHState (Retrieves the file-handle state)

```
#define INCL_DOSFILEMGR

APIRET DosQueryFHState(FileHandle, pFileHandleState);

HFILE   FileHandle;          - File handle
PULONG  pFileHandleState;    - Pointer to variable for file handle state
```

DosQueryFileInfo (Retrieves file information)

```
#define INCL_DOSFILEMGR

APIRET DosQueryFileInfo(FileHandle, FileInfoLevel, pFileInfoBuf,
                        FileInfoBufSize);

HFILE  FileHandle;          - Handle of file
ULONG  FileInfoLevel;       - Level of file information required
PVOID  pFileInfoBuf;        - Pointer to file data buffer
ULONG  FileInfoBufSize;     - Length of file data buffer
```

DosQueryFSAttach (Queries attached file-system information)

```
#define INCL_DOSFILEMGR

APIRET DosQueryFSAttach(pszDeviceName, Ordinal, FSAInfoLevel, pDataBuffer,
                        pDataBufferLen);

PSZ          pszDeviceName;     - Pointer to drive or device
ULONG        Ordinal;           - Index to drive or device
ULONG        FSAInfoLevel;      - Level of information
PFSQBUFFER2  pDataBuffer;       - Pointer to buffer for file-system attributes
PULONG       pDataBufferLen;    - Pointer to data buffer length
```

DosQueryFSInfo (Retrieves file-system information)

```
#define INCL_DOSFILEMGR

APIRET DosQueryFSInfo(DriveNumber, FSInfoLevel, pFSInfoBuf, FSInfoBufSize);

ULONG  DriveNumber;      - Drive number
ULONG  FSInfoLevel;      - Type of information
PVOID  pFSInfoBuf;       - Pointer to buffer for information
ULONG  FSInfoBufSize;    - Length of information buffer
```

DosQueryHType (Determines whether handle is to a file or device)

```
#define INCL_DOSFILEMGR

APIRET DosQueryHType(FileHandle, pHandleType, pFlagWord);

HFILE   FileHandle;     - File handle
PULONG  pHandleType;    - Pointer to variable for handle type
PULONG  pFlagword;      - Pointer to variable for device attribute
```

DosQueryMem (Retrieves page range information)

```
#define INCL_DOSMEMMGR

APIRET DosQueryMem(pBaseAddress, pRegionSize, pAllocationFlags);

PVOID   pBaseAddress;        - Pointer to the memory object
PULONG  pRegionSize;         - Pointer to variable for length of the memory object
PULONG  pAllocationFlags;    - Pointer to variable for attribute flags
```

DosQueryMessageCP (Retrieves list of code pages and language identifiers)

```
#define INCL_DOSMISC

APIRET DosQueryMessageCP(pBufferArea, BufferLength, pszFileName, pBuf);

PCHAR   pBufferArea;     - Pointer to buffer for the list
ULONG   BufferLength;    - Length of buffer
PSZ     pszFileName;     - Pointer to path of the message file
PULONG  pDataLength;     - Receives length of data returned
```

DosQueryModuleHandle (Retrieves a handle to a dynamic-link module)

```
#define INCL_DOSMODULEMGR

APIRET DosQueryModuleHandle(pszModName, pModHandle);

PSZ       pszModName;     - Pointer to dynamic link module name
PHMODULE  pModHandle;     - Pointer to variable receiving module handle
```

DosQueryModuleName (Retrieves path and name of a dynamic link module)

```
#define INCL_DOSMODULEMGR

APIRET DosQueryModuleName(hmodModHandle, BufferLength, pNameBuffer);

HMODULE  hmodModHandle;     - Dynamic link module handle
ULONG    BufferLength;      - Maximum length of bytes in buffer
PCHAR    pNameBuffer;       - Pointer to buffer receiving module name
```

DosQueryMutexSem (Retrieves owner and request count of semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosQueryMutexSem(hmtx, ppidOwner, pptidOwner, pCount);

HMTX    hmtx;          - Handle of the mutex semaphore
PPID    ppidOwner;     - Pointer to the process ID
PTID    pptidOwner;    - Pointer to the thread ID
PULONG  pCount;        - Pointer to the request count of the semaphore
```

DosQueryMuxWaitSem (Retrieves the semaphores in a muxwait semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosQueryMuxWaitSem(hmux, pcSemRec, ppSemRec, pflAttr);

HMUX        hmux;         - Handle of muxwait semaphore
PULONG      pcSemRec;     - Pointer to count of semaphore record entries
PSEMRECORD  ppSemRec;     - Pointer to semaphore record entries
PULONG      pflAttr;      - Pointer to attribute flags
```

DosQueryNPHState (Retrieves handle-state information)

```
#define INCL_DOSNMPIPES

APIRET DosQueryNPHState(hplpeHandle, pPipeHandleState);

HPIPE   hplpeHandle;         - Pipe handle
PULONG  pPipeHandleState;    - Pointer to variable receiving handle state
```

DosQueryNPipeInfo (Retrieves named-pipe information)

```
#define INCL_DOSNMPIPES

APIRET DosQueryNPipeInfo(hplpeHandle, InfoLevel, pInfoBuf, InfoBufSize);

HPIPE  hplpeHandle;     - Pipe handle
ULONG  InfoLevel;       - Level of information to retrieve
PVOID  pInfoBuf;        - Pointer to buffer for information
ULONG  InfoBufSize;     - Length of bytes in buffer
```

DosQueryNPipeSemState (Retrieves named-pipe semaphore information)

```
#define INCL_DOSNMPIPES

APIRET DosQueryNPipeSemState(SemHandle, pInfoBuf, InfoBufLen);

HSEM           SemHandle;      - Semaphore handle
PPIPESEMSTATE  pInforBuf;      - Pointer to buffer receiving information
ULONG          InfoBufLen;     - Buffer size
```

DosQueryPathInfo (Retrieves information about a file or directory)

```
#define INCL_DOSFILEMGR

APIRET DosQueryPathInfo(pszPathName,PathInfoLevel,pPathInfoBuf,PathInfoBufSize);

PSZ    pszPathName;         - Pointer to path name
ULONG  PathInfoLevel;       - Level of information
PVOID  pPathInfoBuf;        - Pointer to buffer for information
ULONG  PathInfoBufSize;     - Length of information buffer
```

DosQueryProcAddr (Retrieves procedure address in a DLL)

```
#define INCL_DOSMODULEMGR

APIRET DosQueryProcAddr(hmodModHandle, Ordinal pszProcName, pProcAddr);

HMODULE  hmodModHandle;     - Handle of module
ULONG    Ordinal;           - Ordinal number
PSZ      pszProcName;       - Pointer to module name string
PFN      pProcAddr;         - Pointer to variable for procedure address
```

DosQueryProcType (Returns the type of the procedure within DLL);

```
#define INCL_DOSMODULEMGR

APIRET DosQueryProcType(hmodModHandle, Ordinal, pszProcName, pProcType);

HMODULE  hmodModuleHandle;     - Handle of module
ULONG    Ordinal;              - Ordinal number of the procedure
PSZ      pszProcName;          - Pointer to the name of procedure
PULONG   pProcType;            - Pointer to the procedure type is returned
```

DosQueryQueue (Retrieves a count of elements in a queue);

```
#define INCL_DOSQUEUES

APIRET DosQueryQueue(QueueHandle, pNumberElements);

HQUEUE  QueueHandle;         - Queue handle
PULONG  pNumberElements;     - Pointer to variable for element count
```

DosQueryResourceSize (Retrieves the size of a resource)

```
#define INCL_DOSMODULEMGR

APIRET DosQueryResourceSize(hmodModHandle, TypeID, NameID, pSize);

HMODULE  hmodModHandle;     - Module handle of the required resource
ULONG    TypeID;            - Type identifier of the resource
ULONG    NameID;            - Name identifier of the resource
PULONG   pSize;             - Pointer to DWORD where resource information returned
```

DosQuerySysInfo (Retrieves system variable values)

```
#define INCL_DOSFILEMGR

APIRET DosQuerySysInfo(StartIndex, LastIndex pdataBuf, DataBufLen);

ULONG  StartIndex;     - Index of first value to return
ULONG  LastIndex;      - Index of last value to return
PVOID  pDataBuf;       - Pointer to buffer receiving information
ULONG  DataBufLen;     - Length of bytes in data buffer
```

DosQueryVerify (Retrieves the write-verification mode)

```
#define INCL_DOSFILEMGR

APIRET DosQueryVerify(pVerifySetting);

PBOOL  pVerifySetting;     - Pointer to verification-mode indicator
```

DosRaiseException (Raises an exception for current thread)

```
#define INCL_DOSEXCEPTIONS

APIRET DosRaiseException(pExceptionReportRecord);

PEXCEPTIONREPORTRECORD  pExceptionReportRecord;     - Exception report record pointer
```

DosRead (Reads from a file or device)

```
#define INCL_DOSFILEMGR

APIRET DosRead(FileHandle, pBufferArea, BufferLength, pBytesRead);

HFILE   FileHandle;       - File handle
PVOID   pBufferArea;      - Pointer to buffer receiving data
ULONG   BufferLength;     - Length of bytes in buffer
PULONG  pBytesRead;       - Pointer to variable for number of bytes read
```

DosReadQueue (Reads an element from a queue)

```
#define INCL_DOSQUEUES

APIRET DosReadQueue(QueueHandle, pRequest, pDataLength, pDataAddress,
                    ElementCode, NoWait, pbElemPriority, SemHandle);

HQUEUE        QueueHandle;       - Handle of queue to read
PREQUESTDATA  pRequest;          - Pointer to structure for PID and request code
PULONG        pDataLength;       - Pointer to length of data
PPVOID        pDataAddress;      - Pointer to buffer for element
ULONG         ElementCode;       - Element number to read
BOOL          NoWait;            - Wait/no wait indicator
PBYTE         pbElemPriority;    - Pointer to variable for priority of element
HEV           SemHandle;         - Semaphore handle
```

DosReleaseMutexSem (Relinquishes ownership of a mutex semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosRequestMutexSem(hmtx);

HMTX  hmtx;     - Semaphore handle
```

DosRequestMutexSem (Requests ownership of a mutex semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosRequestMutexSem(hmtx, Timeout);

HMTX   hmtx;        - Semaphore handle
ULONG  Timeout;     - Milliseconds to wait
```

DosRequestVDD (Request virtual device driver devices)

```
#define INCL_DOSMVDM

APIRET DosRequestVDD(VDDHandle, sgidSessionID, Command, InputBufferLen,
                     pinputBuffer, OutputBufferLen, pOutputBuffer);

HVDD   VDDHandle;          - VDD handle returned by previous DosOpenVDD
SGID   sgidSessionID;      - Identifier of Dos session, or null
ULONG  Command;            - Function code specific to a virtual device
ULONG  InputBufferLen;     - Length of application data in InputBuffer
PVOID  pInputBuffer;       - Address of command-specific info to be sent to VDD
ULONG  OutputBufferLen;    - Length in bytes of output buffer
PVOID  pOutputBuffer;      - Address of buffer where VDD returns information
```

DosResetBuffer (Flushes the file buffers)

```
#define INCL_DOSFILEMGR

APIRET DosResetBuffer(FileHandle);

HFILE  FileHandle;     - File handle
```

DosResetEventSem (Resets an event semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosResetEventSem(hev, pPostCt);

HEV     hev;         - Semaphore handle
PULONG  pPostCt;     - Count of current posts
```

DosResumeThread (Resumes a suspended thread)

```
#define INCL_DOSPROCESS

APIRET DosResumeThread(idThreadID);

TID  idThreadID;     - Identifier of thread to be resumed
```

DosScanEnv (Scans the environment table for a variable)

```
#define INCL_DOSFILEMGR

APIRET DosScanEnv(pszEnvVarName, pszResultPointer);

PSZ  pszEnvVarName;       - Pointer to environment variable name
PSZ  pszResultPointer;    - Pointer to variable for result pointer
```

DosSearchPath (Searches a path for a file)

```
#define INCL_DOSFILEMGR

APIRET DosSearchPath(Control, pszPathRef, pszFileName, pbResultBuffer,
                     ResultBufferLen);

ULONG  Control;            - Control vector
PSZ    pszPathRef;         - Pointer to search path or environment variable
PSZ    pszFileName;        - Pointer to file name
PBYTE  pbResultBuffer;     - Pointer to result buffer
ULONG  ResultBufferLen;    - Length of result buffer
```

DosSelectSession (Switches a child session to the foreground)

```
#define INCL_DOSSESMGR

APIRET DosSelectSession(SessID);

ULONG  SessID;    - Session identifier
```

DosSendSignalException (Sends a Ctrl+C or Ctrl+Break signal exception)

```
#define INCL_DOSEXCEPTIONS

APIRET DosSendSignalException(idpid, Exception);

PID    idpid;         - Identification of the process
ULONG  Exception;     - Number of the signal exception to send
```

DosSetCurrentDir (Defines the current directory)

```
#define INCL_DOSFILEMGR

APIRET DosSetCurrentDir(pszDirName);

PSZ  pszDirName;    - Pointer to directory path
```

DosSetDateTime (Sets the date and time)

```
#define INCL_DOSDATETIME

APIRET DosSetDateTime(pPDateTime);

PDATETIME  pPDateTime;    - Pointer to structure for date and time
```

DosSetDefaultDisk (Selects a default disk drive)

```
#define INCL_DOSFILEMGR

APIRET DosSetDefaultDisk(DriveNumber);

ULONG  DriveNumber;    - Default drive number
```

DosSetExceptionHandler (Registers an exception handler)

```
#define INCL_DOSEXCEPTIONS

APIRET DosSetExceptionHandler(ppERegRec);

PEXCEPTIONREGISTRATIONRECORD  ppERegRec;    - Pointer to exception report record
```

DosSetFHState (Sets the flags for a file handle)

```
#define INCL_DOSFILEMGR

APIRET DosSetFHState(FileHandle, FileHandleState);

HFILE  FileHandle;          - File handle
ULONG  FileHandleState;     - File state flags
```

DosSetFileInfo (Sets file information)

```
#define INCL_DOSFILEMGR

APIRET DosSetFileInfo(FileHandle, FileInfoLevel, pFileInfoBuf, FileInfoBufSize);

HFILE  FileHandle;          - Handle of file about which data sought
ULONG  FileInfoLevel;       - Level of file data required
PVOID  pFileInfoBuf;        - Pointer to file-data buffer
ULONG  FileInfoBufSize;     - Length of file-data buffer
```

DosSetFileLocks (Locks or unlocks a portion of a file)

```
#define INCL_DOSFILEMGR

APIRET DosSetFileLocks(FileHandle, pUnLockRange, pLockRange, Timeout, Flags);

HFILE      FileHandle;      - File handle
PFILELOCK  pUnLockRange;    - Pointer to range to be unlocked
PFILELOCK  pLockRange;      - Pointer to range to be locked
ULONG      Timeout;         - Maximum time in milliseconds
ULONG      Flags;           - Flags that describe the action to be taken
```

DosSetFilePtr (Changes a file pointer)

```
#define INCL_DOSFILEMGR

APIRET DosSetFilePtr(FileHandle, lDistance, MoveType, pNewPointer);

HFILE   FileHandle;     - File handle
LONG    lDistance;      - Distance to move
ULONG   MoveType;       - Method of moving
PULONG  pNewPointer;    - Pointer to new pointer location
```

DosSetFileSize (Changes a file's size)

```
#define INCL_DOSFILEMGR

APIRET DosSetFileSize(FileHandle, FileSize);

HFILE  FileHandle;    - File handle
ULONG  FileSize;      - New size of file
```

DosSetFSInfo (Sets information for file system device)

```
#define INCL_DOSFILEMGR

APIRET DosSetFSInfo(DriveNumber, FSInfoLevel, pFSInfoBuf, FSInfoBufSize);

ULONG  DriveNumber;       - Drive number
ULONG  FSInfoLevel;       - Level of file-system information
PVOID  pFSInfoBuf;        - Pointer to structure for file-system information
ULONG  FSInfoBufSize;     - Length of buffer for file-system information
```

DosSetMaxFH (Sets the maximum number of file handles)

```
#define INCL_DOSFILEMGR

APIRET DosSetMaxFH(NumberHandles);

ULONG  NumberHandles;     - Number of file handles
```

DosSetMem(Changes memory object attributes)

```
#define INCL_DOSMEMMGR

APIRET DosSetMem(pBaseAddress, RegionSize, AttributeFlags);

PVOID  pBaseAddress;      - Pointer to the memory object
ULONG  RegionSize;        - Length of the memory object
ULONG  AttributeFlags;    - Memory object flags
```

DosSetNPHState (Sets mode of named pipe)

```
#define INCL_DOSNMPIPES

APIRET DosSetNPHState(hplpeHandle, PipeHandleState);

HPIPE  hplpeHandle;        - Pipe handle
ULONG  PipeHandleState;    - Named-pipe handle state
```

DosSetNPipeSem (Attaches a share event semaphore to a named pipe)

```
#define INCL_DOSNMPIPES

APIRET DosSetNPipeSem(hplpeHandle, SemHandle, KeyHandle);

HPIPE  hplpeHandle;     - Pipe handle
HSEM   SemHandle;       - Semaphore handle
ULONG  KeyHandle;       - Key value to associate
```

DosSetPathInfo (Sets information for a file or subdirectory)

```
#define INCL_DOSFILEMGR

APIRET DosSetPathInfo(pszPathName, FlagInfoLevel, pFlagInfoBuf, FlagInfoSize,
                      PathInfoFlags);

PSZ    pszPathName;       - Pointer to path
ULONG  FlagInfoLevel;     - Level of information
PVOID  pFlagInfoBuf;      - Pointer to buffer for information
ULONG  FlagInfoSize;      - Length of information buffer
ULONG  PathInfoFlags;     - Information flags
```

DosSetPriority (Sets the priority of a thread or process)

```
#define INCL_DOSPROCESS

APIRET DosSetPriority(Scope, PriorityClass, lPriorityDelta, ID);

ULONG  Scope;             - Scope of change
ULONG  PriorityClass;     - Priority class to set
LONG   lPriorityData;     - Change in priority level
ULONG  ID;                - Process or thread identifier
```

DosSetProcessCp (Sets the code page for a process)

```
#define INCL_DOSNLS

APIRET DosSetProcessCp(CodePage);

ULONG  CodePage;     - Code-page identifier
```

DosSetRelMaxFH (Adjusts the maximum number of file handles)

```
#define INCL_DOSFILEMGR

APIRET DosSetRelMaxFH(pReqCount, pCurMaxFH);

PLONG   pReqCount;     - Pointer to the number to be added
PULONG  pCurMaxFH;     - Pointer to the current number of allocated
```

DosSetSession (Sets the status of a child session)

```
#define INCL_DOSSESMGR

APIRET DosSetSession(dSessID, pStatusData);

ULONG        SessID;         - Session identifier
PSTATUSDATA  pStatusData;    - Pointer to structure for session-status data
```

DosSetSignalExceptionFocus (Initializes for signal exceptions)

```
#define INCL_DOSEXCEPTIONS

APIRET DosSetSignalExceptionFocus(Flag, pTimes);

BOOL    Flag;       - Start/stop flag
PULONG  pTimes;     - Pointer to the number of times it has been called
```

DosSetVerify (Sets write verification on or off)

```
#define INCL_DOSFILEMGR

APIRET DosSetVerify(VerifySetting);

BOOL  VerifySetting;     - Verify mode
```

DosShutdown (Shuts down file systems for power off)

```
#define INCL_DOSFILEMGR

APIRET DosShutdown(Reserved);

ULONG  Reserved;     - Must be zero
```

DosSleep (Temporarily suspends the current thread)

```
#define INCL_DOSDATETIME

APIRET DosSleep(TimeInterval);

ULONG  TimeInterval;     - Number of milliseconds to wait
```

DosStartSession (Starts a session)

```
#define INCL_DOSSESMGR

APIRET DosStartSession(pStartData, pSessID, pPID);

PSTARTDATA  pStartData;     - Pointer to structure with session data
PULONG      pSessID;        - Pointer to variable for session identifier
PPID        pPID;           - Pointer to variable for process identifier
```

DosStartTimer (Starts a periodic timer)

```
#define INCL_DOSDATETIME

APIRET DosStartTimer(TimeInterval, SemHandle, pHandle);

ULONG    TimeInterval;    - Time before semaphore is cleared
HSEM     SemHandle;       - System event semaphore handle
PHTIMER  pHandle;         - Pointer to variable for timer handle
```

DosStopSession (Terminates a session)

```
#define INCL_DOSSESMGR

APIRET DosStopSession(TargetOption, SessID);

ULONG  TargetOption;    - Specifies sessions to end
ULONG  SessID;          - Session identifier
```

DosStopTimer (Stops a timer)

```
#define INCL_DOSDATETIME

APIRET DosStopTimer(htimerHandle);

HTIMER  htimerHandle;    - Timer handle
```

DosSubAlloc (Allocates a memory block from a memory object)

```
#define INCL_DOSMEMMGR

APIRET DosSubAlloc(pOffset, pBlockOffset, Size);

PVOID   pOffset;         - Pointer to the memory object
PPVOID  pBlockOffset;    - Address of the pointer for the memory block
ULONG   Size;            - Length of the memory block
```

DosSubFreeMem (Frees memory allocated by DosSubAlloc)

```
#define INCL_DOSMEMMGR

APIRET DosSubFreeMem(pOffset, pBlockOffset, Size);

PVOID  pOffset;         - Pointer to the memory object
PVOID  pBlockOffset;    - Pointer to the memory block to free
ULONG  Size;            - Length of the memory block
```

DosSubSetMem (Initializes a memory object for suballocation)

```
#define INCL_DOSMEMMGR

APIRET DosSubSetMem(pOffset, Flags, Size);

PVOID  pOffset;     - Pointer to the memory object
ULONG  Flags;       - Indicator flags
ULONG  Size;        - Length of the memory block
```

DosSubUnsetMem (Removes suballocation flags from a memory object)

```
#define INCL_DOSMEMMGR

APIRET DosSubUnsetMem(pOffset);

PVOID  pOffset;     - Pointer to the memory object
```

DosSuspendThread (Suspends a thread)

```
#define INCL_DOSDOSPROCESS

APIRET DosSuspendThread(idThreadID);

TID  idThreadID;     - Identifier of thread to suspend
```

DosTransactNPipe (Performs a transaction on a named pipe)

```
#define INCL_DOSNMPIPES

APIRET DosTransactNPipe(hplpeHandle, pOutBuffer, OutBufferLen, pInBuffer,
                        InBufferLen, pBytesRead)

HPIPE   hplpeHandle;      - Pipe handle
PVOID   pOutBuffer;       - Pointer to buffer with data
ULONG   OutBufferLen;     - Number of bytes in output buffer
PVOID   pInBuffer;        - Pointer to buffer for data
ULONG   InBufferLen;      - Number of bytes in input buffer
PULONG  pBytesRead;       - Pointer to variable for number of bytes read
```

DosUnsetExceptionHandler (Removes an exception handler)

```
#define INCL_DOSEXCEPTIONS

APIRET DosUnsetExceptionHandler(ppERegRec);

PEXCEPTIONREGISTRATIONRECORD  ppERegRec; - Pointer to exception registration record
```

DosUnwindException (Unwinds the exception handlers)

```
#define INCL_DOSEXCEPTIONS

APIRET DosUnwindException(pphandler, pTargetIP, ppERepRec);

PEXCEPTIONREGISTRATIONRECORD  pphandler; - Pointer to exception reg. record
PVOID                         pTargetIP; - Pointer to DosUnwindException branches
PEXCEPTIONREPORTRECORD        ppERepRec; - Optional pointer to an exception record
```

DosWaitChild (Waits for a process to terminate)

```
#define INCL_DOSPROCESS

APIRET DosWaitChild(ActionCode, WaitOption, pReturnCodes, pRetProcessID,
                    idProcessID);

ULONG         ActionCode;       - Indicate process to be terminated
ULONG         WaitOption;       - Wait/no-wait flag
PRESULTCODES  pReturnCodes;     - Pointer to structure receiving result codes
PPID          pRetProcessID;    - Pointer to variable for process identifier
PID           idProcessID;      - Process identifier of process to wait for
```

DosWaitEventSem (Waits for an event semaphore to be posted)

```
#define INCL_DOSSEMAPHORES

APIRET DosWaitEventSem(hev, Timeout);

HEV    hev;         - Semaphore handle
ULONG  Timeout;     - Milliseconds to wait
```

DosWaitMuxWaitSem (Waits for a muxwait semaphore)

```
#define INCL_DOSSEMAPHORES

APIRET DosWaitMuxWaitSem(hmux, Timeout, pUser);

HMUX    hmux;        - Semaphore handle
ULONG   Timeout;     - Milliseconds to wait
PULONG  pUser;       - Pointer to the user record
```

DosWaitNPipe (Waits for a named pipe to become available)

```
#define INCL_DOSDOSNMPIPES

APIRET DosWaitNPipe(pszFileName, Timeout);

PSZ    pszFileName;     - Pointer to pipe name
ULONG  Timeout;         - Timeout value
```

DosWaitThread (Waits for a thread to terminate)

```
#define INCL_DOSPROCESS

APIRET DosWaitThread(pThreadID, WaitOption);

PTID   pThreadID;      - Points to the thread id
ULONG  WaitOption;     - Wait/no-wait flag
```

DosWrite (Writes to a file, device, or pipe)

```
#define INCL_DOSFILEMGR

APIRET DosWrite(FileHandle, pBufferArea, BufferLength, pBytesWritten);

HFILE   FileHandle;        - File handle
PVOID   pBufferArea;       - Pointer to buffer area
ULONG   BufferLength;      - Number of bytes to write
PULONG  pBytesWritten;     - Pointer to variable receiving byte count
```

DosWriteQueue (Writes to a queue)

```
#define INCL_DOSQUEUES

APIRET DosWriteQueue(QueueHandle, Request, DataLength, pDataBuffer, ElemPriority);

HQUEUE  QueueHandle;     - Target-queue handle
ULONG   Request;         - Request/identification data
ULONG   DataLength;      - Number of bytes to write
PVOID   pDataBuffer;     - Pointer to buffer with element to write
ULONG   ElemPriority;    - Priority of element to write
```

Gpi Functions

GpiAnimatePalette (Changes specified colors in a color palette)

```
#define INCL_GPILOGCOLORTABLE

LONG GpiAnimatePalette(hpal, ulFormat, ulStart, ulCount, aulTable);

HPAL    hpal;        - Palette handle
ULONG   ulFormat;    - Format of entries in the table
ULONG   ulStart;     - Starting index
ULONG   ulCount;     - Count of elements in aulTable
PULONG  aulTable;    - Start of application data area
```

GpiAssociate (Associates graphics presentation space with device context)

```
#define INCL_GPICONTROL

BOOL GpiAssociate(hps, hdc);

HPS  hps;    - Presentation-space handle
HDC  hdc;    - Device-context handle
```

GpiBeginArea (Starts construction of an area)

```
#define INCL_GPIPRIMITIVES

BOOL GpiBeginArea(hps, flOptions);

HPS    hps;          - Presentation-space handle
ULONG  flOptions;    - Area option flag
```

GpiBeginElement (Starts an element within a segment)

```
#define INCL_GPISEGEDITING

BOOL GpiBeginElement(hps, lType, pszDesc);

HPS   hps;        - Presentation-space handle
LONG  lType;      - Element type
PSZ   pszDesc;    - Pointer to element description
```

GpiBeginPath (Starts a path)

```
#define INCL_GPIPATHS

BOOL GpiBeginPath(hps, lPath);

HPS   hps;      - Presentation-space handle
LONG  lPath;    - Path identifier
```

GpiBitBlt (Copies bit maps)

```
#define INCL_GPIBITMAPS

LONG GpiBitBlt(hpsTarget, hpsSource, lCount, aptlPoints, lRop, flOptions);

HPS      hpsTarget;     - Target presentation-space handle
HPS      hpsSource;     - Source presentation-space handle
LONG     lCount;        - Number of points in array
PPOINTL  aptlPoints;    - Pointer to array
LONG     lRop;          - Mixing method
ULONG    flOptions;     - Line/column-compression options
```

GpiBox (Draws a rectangular box)

```
#define INCL_GPIPRIMITIVES

LONG GpiBox(hps, lControl, pptlPoint, lHRound, lVRound);

HPS      hps;           - Presentation-space handle
LONG     lControl;      - Fill and outline indicator
PPOINTL  pptlPoint;     - Address of structure for box corners
LONG     lHRound;       - Horizontal length of rounding-ellipse axis
LONG     lVRound;       - Vertical length of rounding-ellipse axis
```

GpiCallSegmentMatrix (Draws a segment using an instance matrix)

```
#define INCL_GPITRANSFORMS

LONG GpiCallSegmentMatrix(hps, lSegment, lCount, pmatlfArray, lOptions);

HPS        hps;            - Presentation-space handle
LONG       lSegment;       - Segment identifier
LONG       lCount;         - Number of matrix elements to examine
PMATRIXLF  pmatlfArray;    - Address of structure for matrix
LONG       lOptions;       - Transformation modifier
```

GpiCharString (Draws a character string at current position)

```
#define INCL_GPIPRIMITIVES

LONG GpiCharString(hps, lCount, pchString);

HPS   hps;          - Presentation-space handle
LONG  lCount;       - Number of characters in string
PCH   pchString;    - Address of string to draw
```

GpiCharStringAt (Draws character string at specified position)

```
#define INCL_GPIPRIMITIVES

LONG GpiCharStringAt(hps, pptlPoint, lCount, pchString);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlPoint;    - Pointer to structure for starting position
LONG     lCount;       - Number of characters in string
PCH      pchString;    - Pointer to string to draw
```

GpiCharStringPos (Draws a character string with formatting)

```
#define INCL_GPIPRIMITIVES

LONG GpiCharStringPos(hps, prclRect, flOptions, lCount, pchString, alAdx);

HPS     hps;          - Presentation-space handle
PRECTL  prclRect;     - Address of structure for rectangle coordinates
ULONG   flOptions;    - Formatting flags
LONG    lCount;       - Number of characters in string
PCH     pchString;    - Pointer to string to draw
PLONG   alAdx;        - Address of array of increment values
```

GpiCharStringPosAt (Draws a character string with formatting)

```
#define INCL_GPIPRIMITIVES

LONG GpiCharStringPosAt(hps, pptlStart, prclRect, flOptions, lCount, pchString,
                        alAdx);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlStart;    - Address of structure for starting position
PRECTL   prclRect;     - Address of structure for rectangle coordinates
ULONG    flOptions;    - Formatting flags
LONG     lCount;       - Number of characters in string
PCH      pchString;    - Address of string to draw
PLONG    alAdx;        - Increment vector
```

GpiCloseFigure (Closes a figure)

```
#define INCL_GPIPATHS

BOOL GpiCloseFigure(hps);

HPS  hps;     - Presentation-space handle
```

GpiCloseSegment (Closes the current segment)

```
#define INCL_GPISEGMENTS

BOOL GpiCloseSegment(hps);

HPS  hps;     - Presentation-space handle
```

GpiCombineRegion (Combines two regions)

```
#define INCL_GPIREGIONS

LONG GpiCombineRegion(hps, hrgnDest, hrgnSrc1, hrgnSrc2, lMode);

HPS   hps;          - Presentation-space handle
HRGN  hrgnDest;     - Handle of destination region
HRGN  hrgnSrc1;     - Handle of first source region
HRGN  hrgnSrc2;     - Handle of second source region
LONG  lMode;        - Combination method
```

GpiComment (Adds a comment to a segment)

```
#define INCL_GPIPRIMITIVES

BOOL GpiComment(hps, lLength, pbData);

HPS    hps;         - Presentation-space handle
LONG   lLength;     - Length of comment string
PBYTE  pbData;      - Address of the comment string
```

GpiConvert (Converts an array of coordinate pairs)

```
#define INCL_GPITRANSFORMS

BOOL GpiConvert(hps, lSrc, lTarg, lCount, aptlPoints);

HPS      hps;           - Presentation-space handle
LONG     lSrc;          - Source coordinate space
LONG     lTarg;         - Target coordinate space
LONG     lCount;        - Number of coordinate pairs in structure
PPOINTL  aptlPoints;    - Address of structure for coordinate pairs
```

GpiConvertWithMatrix (Converts array between coordinate systems)

```
#define INCL_GPITRANSFORMS

BOOL GpiConvertWithMatrix(hps, lCount, aptlPoints, lCount, pmatlfArray);

HPS        hps;            - Presentation-space handle
LONG       lCount;         - Number of points in array
PPOINTL    aptlPoints;     - Pointer to array of structures for points
LONG       lCount;         - Number of elements in array to examine
PMATRIXLF  pmatlfArray;    - Pointer to structure for transformation matrix
```

GpiCopyMetaFile (Copies a metafile)

```
#define INCL_GPIMETAFILES

HMF GpiCopyMetaFile(hmf);

HMF  hmf;    - Handle of source metafile
```

GpiCorrelateChain (Correlates a chain)

```
#define INCL_GPICORRELATION

LONG GpiCorrelateChain(hps, lType, pptlPick, lMaxHits, lMaxDepth, alSegTag);

HPS      hps;          - Presentation-space handle
LONG     lType;        - Segment type
PPOINTL  pptlPick;     - Address of structure for aperture center
LONG     lMaxHits;     - Maximum number of hits
LONG     lMaxDepth;    - Maximum number of segment/tag pairs to return
PLONG    alSegTag;     - Address of array of segment and tag identifiers
```

GpiCorrelateFrom (Performs a correlation operation)

```
#define INCL_GPICORRELATION

LONG GpiCorrelateFrom(hps, lFirstSegment, lLastSegment, lType, pptl, lMaxHits,
                      lMaxDepth, alSegTag);

HPS      hps;              - Presentation-space handle
LONG     lFirstSegment;    - First segment to correlate
LONG     lLastSegment;     - Last segment to correlate
LONG     lType;            - Segment type
PPOINTL  pptl;             - Address of structure for aperture center
LONG     lMaxHits;         - Maximum number of hits
LONG     lMaxDepth;        - Maximum number of segment/tag pairs to return
PLONG    alSegTag;         - Address of array of segment/tag identifiers
```

GpiCorrelateSegment (Correlates a segment)

```
#define INCL_GPICORRELATION

LONG GpiCorrelateSegment(hps, lSegment, lType, pptlPick, lMaxHits, lMaxDepth,
                         alSegTag);

HPS      hps;          - Presentation-space handle
LONG     lSegment;     - Segment to correlate
LONG     lType;        - Segment type
PPOINTL  pptlPick;     - Address of structure for aperture center
LONG     lMaxHits;     - Maximum number of hits
LONG     lMaxDepth;    - Maximum number of segment|tag pairs to return
PLONG    alSegTag;     - Address of array of segment and tag identifiers
```

GpiCreateBitmap (Creates a bit map)

```
#define INCL_GPIBITMAPS

HBITMAP GpiCreateBitmap(hps, pbmpNew, flOptions, pbInitData, pbmInfoData);

HPS                  hps;          - Presentation-space handle
PBITMAPINFOHEADER2   pbmpNew;      - Address of structure for format data
ULONG                flOptions;    - Options
PBYTE                pbInitData;   - Address of buffer of image data
PBITMAPINFO2         pbmInfoData;  - Address of structure for color and format
```

GpiCreateLogColorTable (Creates a logical color table)

```
#define INCL_GPILOGCOLORTABLE

BOOL GpiCreateLogColorTable(hps, flOptions, lFormat, lStart, lCount, alTable);

HPS    hps;          - Presentation-space handle
ULONG  flOptions;    - Options
LONG   lFormat;      - Format of entries
LONG   lStart;       - Starting index
LONG   lCount;       - Number of entries in table
PLONG  alTable;      - Address of array for table
```

GpiCreateLogFont (Creates a logical font)

```
#define INCL_GPILCIDS

LONG GpiCreateLogFont(hps, pName, lLcid, pAttrs);

HPS      hps;        - Presentation-space handle
PSTR8    pName;      - Address of logical-font name
LONG     lLcid;      - Local identifier
PFATTRS  pAttrs;     - Address of structure for font attributes
```

GpiCreatePalette (Creates a color palette)

```
#define INCL_GPILOGCOLORTABLE

HPAL GpiCreatePalette(hab, flOptions, flFormat, lStart, lCount, alTable);

HAB    hab;          - Anchor-block handle
ULONG  flOptions;    - Options
LONG   flFormat;     - Format of table entries
LONG   lCount;       - Count of elements in table
PLONG  alTable;      - Address of color table
```

GpiCreatePS (Creates a presentation space)

```
#define INCL_GPICONTROL

HPS GpiCreatePS(hab, hdc, psizlSize, flOptions);

HAB     hab;          - Anchor-block handle
HDC     hdc;          - Device-context handle
PSIZEL  psizlSize;    - Address of structure for page size
ULONG   flOptions;    - Presentation-space options
```

GpiCreateRegion (Creates a region)

```
#define INCL_GPIREGIONS

HRGN GpiCreateRegion(hps, lCount, arclRectangles);

HPS     hps;              - Presentation-space handle
LONG    lCount;           - Number of rectangles
PRECTL  arclRectangles;   - Address of structure for rectangles
```

GpiDeleteBitmap (Deletes a bit map)

```
#define INCL_GPIBITMAPS

BOOL GpiDeleteBitmap(hbm);

HBITMAP  hbm;     - Bitmap handle
```

GpiDeleteElement (Deletes an element)

```
#define INCL_GPISEGEDITING

BOOL GpiDeleteElement(hps);

HPS  hps;     - Presentation-space handle
```

GpiDeleteElementRange (Deletes an element range)

```
#define INCL_GPISEGEDITING

BOOL GpiDeleteElementRange(hps, lFirstElement, lLastElement);

HPS   hps;              - Presentation-space handle
LONG  lFirstElement;    - First element
LONG  lLastElement;     - Last element
```

GpiDeleteElementsBetweenLabels (Deletes the elements between two labels)

```
#define INCL_GPISEGEDITING

BOOL GpiDeleteElementsBetweenLabels(hps, lFirstLabel, lLastLabel);

HPS   hps;            - Presentation-space handle
LONG  lFirstLabel;    - Label of first element
LONG  lLastLabel;     - Label of last element
```

GpiDeleteMetaFile (Deletes a metafile)

```
#define INCL_GPIMETAFILES

BOOL GpiDeleteMetaFile(hmf);

HMF  hmf;     - Metafile handle
```

GpiDeletePalette (Deletes a color palette)

```
#define INCL_GPILOGCOLORTABLE

BOOL GpiDeletePalette(hpal);

HPAL  hpal;     - Palette handle
```

GpiDeleteSegment (Deletes a retained segment)

```
#define INCL_GPISEGMENTS

BOOL GpiDeleteSegment(hps, lSegid);

HPS   hps;       - Presentation-space handle
LONG  lSegid;    - Identifier of segment to delete
```

GpiDeleteSegments (Deletes all segments)

```
#define INCL_GPIsEGMENTS

BOOL GpiDeleteSegments(hps, lFirstSegment, lLastSegment);

HPS   hps;               - Presentation-space handle
LONG  lFirstSegment;     - Identifier of first segment
LONG  lLastSegment;      - Identifier of last segment
```

GpiDeleteSetId (Deletes a logical font or bit-map tag)

```
#define INCL_GPILCIDS

BOOL GpiDeleteSetId(hps, lLcid);

HPS   hps;       - Presentation-space handle
LONG  lLcid;     - Local identifier for font or bit map
```

GpiDestroyPS (Destroys a presentation space)

```
#define INCL_GPICONTROL

BOOL GpiDestroyPS(hps);

HPS  hps;     - Presentation-space handle
```

GpiDestroyRegion (Destroys a region)

```
#define INCL_GPIREGIONS

BOOL GpiDestroyRegion(hps, hrgn);

HPS   hps;      - Presentation-space handle
HRGN  hrgn;     - Handle of region to destroy
```

GpiDrawBits (Draws a rectangle of bits)

```
#define INCL_GPIBITMAPS

LONG GpiDrawBits(hps, pBits, pbmInfoTable, lCount, aptlPoints, lRop, flOptions);

HPS           hps;             - Target presentation-space handle
PVOID         pBits;           - Address of source bits
PBITMAPINFO2  pbmInfoTable;    - Address of bit-map information table
LONG          lCount;          - Count of points
PPOINTL       aptlPoints;      - Address of structure for points
LONG          lRop;            - Mixing function
ULONG         flOptions;       - Options
```

GpiDrawChain (Draws a picture chain)

```
#define INCL_GPISEGMENTS

BOOL GpiDrawChain(hps);

HPS  hps;     - Presentation-space handle
```

GpiDrawDynamics (Redraws dynamic segments)

```
#define INCL_GPISEGMENTS

BOOL GpiDrawDynamics(hps);

HPS  hps;     - Presentation-space handle
```

GpiDrawFrom (Draws a section of a picture chain)

```
#define INCL_GPISEGMENTS

BOOL GpiDrawFrom(hps, lFirstSegment, lLastSegment);

HPS   hps;              - Presentation-space handle
LONG  lFirstSegment;    - First chain segment to draw
LONG  lLastSegment;     - Last chain segment to draw
```

GpiDrawSegment (Draws a specified segment)

```
#define INCL_GPISEGMENTS

BOOL GpiDrawSegment(hps, lSegment);

HPS   hps;          - Presentation-space handle
LONG  lSegment;     - Identifier of segment to draw
```

GpiElement (Draws an element)

```
#define INCL_GPISEGEDITING

LONG GpiElement(hps, lType, pszDesc, lLength, pbData);

HPS    hps;         - Presentation-space handle
LONG   lType;       - Element type
PSZ    pszDesc;     - Address of element descriptor
LONG   lLength;     - Length in bytes of buffer for graphics orders
PBYTE  pbData;      - Address of buffer for graphics orders
```

GpiEndArea (Ends an area bracket)

```
#define INCL_GPIPRIMITIVES

LONG GpiEndArea(hps);

HPS  hps;     - Presentation-space handle
```

GpiEndElement (Ends an element bracket)

```
#define INCL_GPISEGEDITING

BOOL GpiEndElement(hps);

HPS  hps;     - Presentation-space handle
```

GpiEndPath (Ends a path bracket)

```
#define INCL_GPIPATHS

BOOL GpiEndPath(hps);

HPS  hps;     - Presentation-space handle
```

GpiEqualRegion (Checks two regions for equality)

```
#define INCL_GPIREGIONS

LONG GpiEqualRegion(hps, hrgnSrc1, hrgnSrc2);

HPS   hps;          - Presentation-space handle
HRGN  hrgnSrc1;     - Handle of the first region
HRGN  hrgnSrc2;     - Handle of the second region
```

GpiErase (Clears the output display)

```
#define INCL_GPICONTROL

BOOL GpiErase(hps);

HPS  hps;     - Presentation-space handle
```

GpiErrorSegmentData (Returns an error location in a segment)

```
#define INCL_GPICONTROL

LONG GpiErrorSegmentData(hps, plSegment, plContext);

HPS    hps;          - Presentation-space handle
PLONG  plSegment;    - Address of segment identifier
PLONG  plContext;    - Address of variable for error type
```

GpiExcludeClipRectangle (Excludes a rectangle from the clip region)

```
#define INCL_GPIREGIONS

LONG GpiExcludeClipRectangle(hps, prclRect);

HPS     hps;         - Presentation-space handle
PRECTL  prclRect;    - Address of structure for rectangle coordinates
```

GpiFillPath (Draws the interior of a path)

```
#define INCL_GPIPATHS

LONG GpiFillPath(hps, lPath, lOptions);

HPS   hps;          - Presentation-space handl
LONG  lPath;        - Identifier of path
LONG  lOptions;     - Fill mode
```

GpiFloodFill (Fills an area defined by a specified color)

```
#define INCL_GPIBITMAPS

BOOL GpiFloodFill(hps, lOptions, lColor);

HPS   hps;          - Presentation-space handle
LONG  lOptions;     - Options
LONG  lColor;       - Color
```

GpiFrameRegion (Draws a frame inside a region)

```
#define INCL_GPIREGIONS

LONG GpiFrameRegion(hps, hrgn, psizlThickness);

HPS     hps;              - Presentation-space handle
HRGN    hrgn;             - Region handle
PSIZEL  psizlThickness;   - Pointer to structure for frame thickness
```

GpiFullArc (Creates a full arc)

```
#define INCL_GPIPRIMITIVES

LONG GpiFullArc(hps, lControl, fxMultiplier);

HPS    hps;             - Presentation-space handle
LONG   lControl;         - Fill and outline indicator
FIXED  fxMultiplier;    - Arc-size multiplier
```

GpiGetData (Get graphics-order data from a segment)

```
#define INCL_GPISEGMENTS

LONG GpiGetData(hps, lSegid, plOffset, lFormat, lLength, pbData);

HPS    hps;         - Presentation-space handle
LONG   lSegid;      - Segment identifier
PLONG  plOffset;    - Address of variable for segment offset
LONG   lFormat;     - Conversion type
LONG   lLength;     - Length in bytes of the data buffer
PBYTE  pbData;      - Address of buffer for data
```

GpiImage (Draws an image)

```
#define INCL_GPIPRIMITIVES

LONG GpiImage(hps, lFormat, psizlImageSize, lLength, pbData);

HPS     hps;              - Presentation-space handle
LONG    lFormat;          - Image data format
PSIZEL  psizlImageSize;   - Address of structure for image width and height
LONG    lLength;          - Length in bytes of the image data
PBYTE   pbData;           - Address of image data
```

GpiIntersectClipRectangle (Sets a clip region from an intersection)

```
#define INCL_GPIREGIONS

LONG GpiIntersectClipRectangle(hps, prclRect);

HPS     hps;         - Presentation-space handle
PRECTL  prclRect;    - Address of structure for rectangle coordinates
```

GpiLabel (Creates a label element)

```
#define INCL_GPISEGEDITING

BOOL GpiLabel(hps, lLabel);

HPS   hps;       - Presentation-space handle
LONG  lLabel;    - Label
```

GpiLine (Draws a line)

```
#define INCL_GPIPRIMITIVES

LONG GpiLine(hps, pptlEndPoint);

HPS      hps;              - Presentation-space handle
PPOINTL  pptlEndPoint;     - Address of structure for the end point
```

GpiLoadBitmap (Loads a bit map from a resource)

```
#define INCL_GPIBITMAPS

HBITMAP GpiLoadBitmap(hps, Resource, idBitmap, lWidth, lHeight);

HPS      hps;          - Presentation-space handle
HMODULE  Resource;     - Resource identity
ULONG    idBitmap;     - Bit-map identifier
LONG     lWidth;       - Width in pels of the bit map
LONG     lHeight;      - Height in pels of the bit map
```

GpiLoadFonts (Loads fonts from a resource file)

```
#define INCL_GPILCIDS

BOOL GpiLoadFonts(hab, pszFileName);

HAB  hab;              - Anchor-block handle
PSZ  pszFileName;      - Pointer to filename
```

GpiLoadMetaFile (Loads data from a file into a metafile)

```
#define INCL_GPIMETAFILES

HMF GpiLoadMetaFile(hab, pszFileName);

HAB  hab;              - Anchor-block handle
PSZ  pszFileName;      - Address of filename of metafile
```

GpiLoadPublicFonts (Loads fonts from specified resource file)

```
#define INCL_GPILCIDS

BOOL GpiLoadPublicFonts(hab, pszFileName);

HAB  hab;              - Anchor-block handle
PSZ  pszFileName;      - Pointer to filename
```

GpiMarker (Draws a marker)

```
#define INCL_GPIPRIMITIVES

LONG GpiMarker(hps, pptlPoint);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlPoint;    - Address of structure for marker position
```

GpiModifyPath (Modifies a path)

```
#define INCL_GPIPATHS

BOOL GpiModifyPath(hps, lPath, lMode);

HPS   hps;       - Presentation-space handle
LONG  lPath;     - Path identifier
LONG  lMode;     - Modification options
```

GpiMove (Moves current position to a specified point)

```
#define INCL_GPIPRIMITIVES

BOOL GpiMove(hps, pptlPoint);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlPoint;    - Address of structure for new position
```

GpiOffsetClipRegion (Moves the clip region)

```
#define INCL_GPIREGIONS

LONG GpiOffsetClipRegion(hps, pptlPoint);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlPoint;    - Address of structure for offset increments
```

GpiOffsetElementPointer (Sets the element pointer)

```
#define INCL_GPISEGEDITING

BOOL GpiOffsetElementPointer(hps, lOffset);

HPS   hps;        - Presentation-space handle
LONG  lOffset;    - Offset to add to element pointer
```

GpiOffsetRegion (Moves a region)

```
#define INCL_GPIREGIONS

BOOL GpiOffsetRegion(hps, hrgn, pptlOffset);

HPS      hps;           - Presentation-space handle
HRGN     hrgn;          - Region handle
PPOINTL  pptlOffset;    - Address of structure for offset increments
```

GpiOpenSegment (Opens a segment)

```
#define INCL_GPISEGMENTS

BOOL GpiOpenSegment(hps, lSegment);

HPS   hps;         - Presentation-space handle
LONG  lSegment;    - Segment identifier
```

GpiOutlinePath (Draws the outline of a path)

```
#define INCL_GPIPATHS

LONG GpiOutlinePath(hps, lPath, lOptions);

HPS   hps;          - Presentation-space handle
LONG  lPath;        - Identifies path to be outlined
LONG  lOptions;     - Reserved, must be zero
```

GpiPaintRegion (Paints a region)

```
#define INCL_GPIREGIONS

LONG GpiPaintRegion(hps, hrgn);

HPS   hps;      - Presentation-space handle
HRGN  hrgn;     - Region handle
```

GpiPartialArc (Draws a partial arc)

```
#define INCL_GPIPRIMITIVES

LONG GpiPartialArc(hps, pptlCenter, fxMultiplier, fxStartAngle, fxSweepAngle);

HPS      hps;              - Presentation-space handle
PPOINTL  pptlCenter;       - Address of structure for center point
FIXED    fxMultiplier;     - Arc-size multiplier
FIXED    fxStartAngle;     - Start angle of arc
FIXED    fxSweepAngle;     - Sweep angle of arc
```

GpiPathToRegion (Converts a path into a region)

```
#define INCL_GPIPATHS

HRGN GpiPathToRegion(hps, lPath, flOptions);

HPS    hps;          - Presentation-space handle
LONG   lPath;        - Local identifier
ULONG  flOptions;    - Options
```

GpiPlayMetaFile (Plays a metafile)

```
#define INCL_GPIMETAFILES

LONG GpiPlayMetaFile(hps, hmf, lCount1, alOptarray, plSegCount, lCount2, pszDesc);

HPS    hps;           - Presentation-space handle
HMF    hmf;           - Metafile handle
LONG   lCount1;       - Number of elements in array
PLONG  alOptarray;    - Address of array of load options
PLONG  plSegCount;    - Address of count of renumbered segments
LONG   lCount2;       - Number of bytes in record
PSZ    pszDesc;       - Address of buffer for descriptive record
```

GpiPointArc (Draws an arc through three points)

```
#define INCL_GPIPRIMITIVES

LONG GpiPointArc(hps, aptlPoints);

HPS      hps;           - Presentation-space handle
PPOINTL  aptlPoints;    - Address of structure for points
```

GpiPolyFillet (Draws a curve)

```
#define INCL_GPIPRIMITIVES

LONG GpiPolyFillet(hps, lCount, aptlPoints);

HPS      hps;           - Presentation-space handlet
LONG     lCount;        - Number of points in array lines
PPOINTL  aptlPoints;    - Address of array of structures for points
```

GpiPolyFilletSharp (Draws a fillet)

```
#define INCL_GPIPRIMITIVES

LONG GpiPolyFilletSharp(hps, lCount, aptlPoints, afxSharpness);

HPS      hps;             - Presentation-space handle
LONG     lCount;          - Number of points
PPOINTL  aptlPoints;      - Array of structures for points
PFIXED   afxSharpness;    - Array of structures for sharpness values
```

GpiPolygons (Draws a set of closed polygons)

```
#define INCL_GPIPOLYGON

LONG GpiPolygons(hps, ulCount, paplgn, flOptions, flmodel);

HPS       hps;           - Presentation-space handle
ULONG     ulCount;       - Number of polygons
PPOLYGON  paplgn;        - Array of polygons
LONG      flOptions;     - Drawing options
LONG      flmodel;       - Drawing model
```

GpiPolyLine (Draws straight lines)

```
#define INCL_GPIPRIMITIVES

LONG GpiPolyLine(hps, lCount, aptlPoints);

HPS      hps;           - Presentation-space handle
LONG     lCount;        - Number of points in array
PPOINTL  aptlPoints;    - Address of array of structures for points
```

GpiPolyLineDisjoint (Draws a series of disjoint straight lines)

```
#define INCL_GPIPRIMITIVES

LONG GpiPolyLineDisjoint(hps, lCount, aptlPoints);

HPS      hps;          - Presentation-space handle
LONG     lCount;       - Number of points
PPOINTL  aptlPoints;    - Array of points
```

GpiPolyMarker (Draws a marker)

```
#define INCL_GPIPRIMITIVES

LONG GpiPolyMarker(hps, lCount, aptlPoints);

HPS      hps;           - Presentation-space handle
LONG     lCount;        - Number of points
PPOINTL  aptlPoints;    - Address of array of structures for point
```

GpiPolySpline (Draws Bezier splines)

```
#define INCL_GPIPRIMITIVES

LONG GpiPolySpline(hps, lCount, aptlPoints);

HPS      hps;           - Presentation-space handle
LONG     lCount;        - Number of points in array
PPOINTL  aptlPoints;    - Address of array of structures for points
```

GpiPop (Restores one or more primitive attributes)

```
#define INCL_GPIPRIMITIVES

BOOL GpiPop(hps, lCount);

HPS   hps;       - Presentation-space handle
LONG  lCount;    - Number of attributes to restore
```

GpiPtInRegion (Determines whether a point is in a region)

```
#define INCL_GPIREGIONS

LONG GpiPtInRegion(hps, hrgn, pptlPoint);

HPS      hps;          - Presentation-space handle
HRGN     hrgn;         - Region handle
PPOINTL  pptlPoint;    - Address of structure for point
```

GpiPtVisible (Determines whether a point is visible)

```
#define INCL_GPIPRIMITIVES

LONG GpiPtVisible(hps, pptlPoint);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlPoint;    - Address of structure for point
```

GpiPutData (Draws graphics orders from a buffer)

```
#define INCL_GPISEGMENTS

LONG GpiPutData(hps, lFormat, plLength, pbData);

HPS    hps;         - Presentation-space handle
LONG   lFormat;     - Coordinate type
PLONG  plLength;    - Address of variable for length of order data
PBYTE  pbData;      - Address of buffer for order data
```

GpiQueryArcParams (Retrieves the current arc parameters)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryArcParams(hps, parcpArcParams);

HPS         hps;              - Presentation-space handle
PARCPARAMS  parcpArcParams;   - Address of structure for arc parameters
```

GpiQueryAttrMode (Retrieves the current attribute mode)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryAttrMode(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryAttrs (Retrieves attributes for a primitive type)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryAttrs(hps, lPrimType, flAttrsMask, ppbunAttrs);

HPS      hps;            - Presentation-space handle
LONG     lPrimType;      - Primitive type
ULONG    flAttrsMask;    - Attribute mask
PBUNDLE  ppbunAttrs;     - Address of buffer for nondefault attributes
```

GpiQueryBackColor (Retrieves the current background color)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryBackColor(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryBackMix (Retrieves the current background mix mode)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryBackMix(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryBitmapBits (Copies bit-map image data to a buffer)

```
#define INCL_GPIBITMAPS

LONG GpiQueryBitmapBits(hps, lScanStart, lScans, pbBuffer, pbmi2InfoTable);

HPS           hps;              - Presentation-space handle
LONG          lScanStart;       - Number for first scan line to retrieve
LONG          lScans;           - Number of scan lines to retrieve
PBYTE         pbBuffer;         - Address of buffer for bit-map image data
PBITMAPINFO2  pbmi2InfoTable;   - Address of structure for bit-map info
```

GpiQueryBitmapDimension (Retrieves the dimensions of a bit map)

```
#define INCL_GPIBITMAPS

BOOL GpiQueryBitmapDimension(hbm, psizlBitmapDimension);

HBITMAP  hbm              - Bit-map handle
PSIZEL   psizlDimension   - Address of structure for bit-map size info
```

GpiQueryBitmapHandle (Retrieves the handle to a bit map)

```
#define INCL_GPIBITMAPS

HBITMAP GpiQueryBitmapHandle(hps, lLcid);

HPS   hps;      - Presentation-space handle
LONG  lLcid;    - Local identifier
```

GpiQueryBitmapInfoHeader (Retrieves information about a bit map)

```
#define INCL_GPIBITMAPS

BOOL GpiQueryBitmapInfoHeader(hbm, pbmp2Data);

HBITMAP             hbm;          - Bit-map handle
PBITMAPINFOHEADER2  pbmp2Data;    - Address of structure for bit-map information
```

GpiQueryBitmapParameters (Retrieves bit-map parameters)

```
#define INCL_GPIBITMAPS

BOOL GpiQueryBitmapParameters(hbm, pbmpData);

HBITMAP            hbm;         - Bit-map handle
PBITMAPINFOHEADER  pbmpData;    - Address of structure for bit-map info
```

GpiQueryBoundaryData (Retrieves boundary data)

```
#define INCL_GPICORRELATION

BOOL GpiQueryBoundaryData(hps, prclBoundary);

HPS     hps;            - Presentation-space handle
PRECTL  prclBoundary;   - Address of structure for boundary data
```

GpiQueryCharAngle (Retrieves the character-angle attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryCharAngle(hps, pgradlAngle);

HPS         hps;            - Presentation-space handle
PGRADIENTL  pgradlAngle;    - Address of structure for baseline-angle point
```

GpiQueryCharBox (Retrieves the character-box attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryCharBox(hps, psizfxSize);

HPS     hps;           - Presentation-space handle
PSIZEF  psizfxSize;    - Address of structure for character-box size
```

GpiQueryCharBreakExtra (Retrieves character-break-extra attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryCharBreakExtra(hps, pfxBreakExtra);

HPS     hps;              - Presentation-space handle
PFIXED  pfxBreakExtra;    - Address of character-break-extra value
```

GpiQueryCharDirection (Retrieves the character-direction attribute)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryCharDirection(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryCharExtra (Retrieves character-extra attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryCharExtra(hps, pfxExtra);

HPS     hps;         - Presentation-space handle
PFIXED  pfxExtra;    - Address of character-extra attribute
```

GpiQueryCharMode (Retrieves the character-mode attribute)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryCharMode(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryCharSet (Retrieves the character-set identifier)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryCharSet(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryCharShear (Retrieves the character-shear angle)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryCharShear(hps, pptlShear);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlShear;    - Address of structure for shear-vector point
```

GpiQueryCharStringPos (Retrieves the positions of characters)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryCharStringPos(hps, flOptions, lCount, pchString, alXincrements,
                           aptlPositions);

HPS      hps;              - Presentation-space handle
ULONG    flOptions;        - Option flags
LONG     lCount;           - Length of the string
PCH      pchString;        - Address of string to examine
PLONG    alXincrements;    - Address of array for increment values
PPOINTL  aptlPositions;    - Address of array of structures for points
```

GpiQueryCharStringPosAt (Retrieves the character positions at a point)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryCharStringPosAt(hps, pptlStart, flOptions, lCount, pchString,
                             alXincrements, aptlPositions);

HPS      hps;              - Presentation-space handle
PPOINTL  pptlStart    ;    - Address of structure for starting point
ULONG    flOptions;        - Option flag
LONG     lCount;           - Length of the string
PCH      pchString;        - Address of string to examine
PLONG    alXincrements;    - Address of array for increment values
PPOINTL  aptlPositions;    - Address of array of structures for points
```

GpiQueryClipBox (Retrieves a clip-box rectangle)

```
#define INCL_GPIREGIONS

LONG GpiQueryClipBox(hps, prclBound);

HPS     hps;          - Presentation-space handle
PRECTL  prclBound;    - Address of structure for clip box
```

GpiQueryClipRegion (Retrieves the handle to a clip region)

```
#define INCL_GPIREGIONS

HRGN GpiQueryClipRegion(hps);

HPS  hps;     - Presentation-space handle
```

GpiQueryColor (Retrieves the line-color attribute)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryColor(hps);

HPS  hps;     - Presentation-space handle
```

GpiQueryColorData (Retrieves color-table data)

```
#define INCL_GPILOGCOLORTABLE

BOOL GpiQueryColorData(hps, lCount, alArray);

HPS    hps;        - Presentation-space handle
LONG   lCount;     - Number of elements
PLONG  alArray;    - Three-element array
```

GpiQueryColorIndex (Retrieves the color index)

```
#define INCL_GPILOGCOLORTABLE

LONG GpiQueryColorIndex(hps, ulOptions, lRgbColor);

HPS    hps;          - Presentation-space handle
ULONG  ulOptions;    - Must be zero
LONG   lRgbColor;    - RGB color value
```

GpiQueryCp (Retrieves a code-page identifier)

```
#define INCL_GPILCIDS

USHORT  GpiQueryCp(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryCurrentPosition (Retrieves the current position)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryCurrentPosition(hps, pptlPoint);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlPoint;    - Address of structure for current position
```

GpiQueryDefArcParams (Retrieves the default arc parameters)

```
#define INCL_GPIDEFAULTS

BOOL GpiQueryDefArcParams(hps, parcpArcParams);

HPS         hps;              - Presentation-space handle
PARCPARAMS  parcpArcParams;   - Pointer to structure for arc parameters
```

GpiQueryDefAttrs (Retrieves the default attributes)

```
#define INCL_GPIDEFAULTS

BOOL GpiQueryDefAttrs(hps, lPrimType, flAttrMask, pbunAttrs);

HPS      hps;          - Presentation-space handle
LONG     lPrimType;    - Primitive type
ULONG    flAttrMask;   - Attributes mask
PBUNDLE  pbunAttrs;    - Pointer to structure for default attributes
```

GpiQueryDefaultViewMatrix (Retrieves the default viewing matrix)

```
#define INCL_GPITRANSFORMS

BOOL GpiQueryDefaultViewMatrix(hps, lCount, pmatlfArray);

HPS        hps;            - Presentation-space handle
LONG       lCount;         - Number of elements
PMATRIXLF  pmatlfArray;    - Address of structure for transformation matrix
```

GpiQueryDefCharBox (Retrieves the size of the default char box)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryDefCharBox(hps, psizlSize);

HPS     hps;          - Presentation-space handle
PSIZEL  psizlSize;    - Address of structure for character-box size
```

GpiQueryDefTag (Retrieves the default tag for a primitive)

```
#define INCL_GPIDEFAULTS

BOOL GpiQueryDefTag(hps, plTag);

HPS    hps;      - Presentation-space handle
PLONG  plTag;    - Pointer to tag
```

GpiQueryDefViewingLimits (Retrieves the default viewing limits)

```
#define INCL_GPIDEFAULTS

BOOL GpiQueryDefViewingLimits(hps, prclLimits);

HPS     hps;           - Presentation-space handle
PRECTL  prclLimits;    - Pointer to structure for viewing limits
```

GpiQueryDevice (Retrieves a device context from a PS)

```
#define INCL_GPICONTROL

HDC GpiQueryDevice(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryDeviceBitmapFormats (Retrieves bit-map formats)

```
#define INCL_GPIBITMAPS

BOOL GpiQueryDeviceBitmapFormats(hps, lCount, alArray);

HPS    hps;        - Presentation-space handle
LONG   lCount;     - Number of elements
PLONG  alArray;    - Array of elements
```

GpiQueryDrawControl (Checks for a drawing control)

```
#define INCL_GPICONTROL

LONG GpiQueryDrawControl(hps, lControl);

HPS   hps;          - Presentation-space handle
LONG  lControl;     - Drawing-control flag
```

GpiQueryDrawingMode (Retrieves the drawing mode)

```
#define INCL_GPICONTROL

LONG GpiQueryDrawingMode(hps);

HPS  hps;     - Presentation-space handle
```

GpiQueryEditMode (Retrieves the current editing mode)

```
#define INCL_GPISEGEDITING

LONG GpiQueryEditMode(hps);

HPS  hps;     - Presentation-space handle
```

GpiQueryElement (Retrieves element content data)

```
#define INCL_GPISEGEDITING

LONG GpiQueryElement(hps, lOff, lMaxLength, pbData);

HPS    hps;           - Presentation-space handle
LONG   lOff;          - Offset to first byte to copy
LONG   lMaxLength;    - Size of the buffer
PBYTE  pbData;        - Address of buffer for graphics-order data
```

GpiQueryElementPointer (Retrieves the current element pointer)

```
#define INCL_GPISEGEDITING

LONG GpiQueryElementPointer(hps);

HPS  hps;     - Presentation-space handle
```

GpiQueryElementType (Retrieves information about an element type)

```
#define INCL_GPISEGEDITING

LONG GpiQueryElementType(hps, plType, lLength, pszData);

HPS    hps;         - Presentation-space handle
PLONG  plType;      - Address of variable for type
LONG   lLength;     - Length in bytes of buffer
PSZ    pszData;     - Address of buffer for description
```

GpiQueryFaceString (Retrieves a compound face name for a font)

```
#define INCL_GPILCIDS

ULONG GpiQueryFaceString(hps, pszFamilyName, pfndFaceAttrs, lLength,
                         pszCompoundFaceName);

HPS            hps;                  - Presentation-space handle
PSZ            pszFamilyName;        - Pointer to simple face name
PFACENAMEDESC  pfndFaceAttrs;        - Pointer to structure with face description
LONG           lLength;              - Size of buffer for compound face name
PSZ            pszCompoundFaceName;  - Pointer to compound face name
```

GpiQueryFontAction (Checks whether available fonts have changed)

```
#define INCL_GPILCIDS

ULONG GpiQueryFontAction(hab, flOptions);

HAB    hab;          - Anchor-block handle
ULONG  flOptions;    - Options
```

GpiQueryFontFileDescriptions (Retrieves font-file descriptions)

```
#define INCL_GPILCIDS

LONG GpiQueryFontFileDescriptions(hab, pszFileName, pcCount, affdescsNames);

HAB       hab;              - Anchor-block handle
PSZ       pszFileName;      - Address of the font-resource filename
PLONG     pcCount;          - Address of variable with number of fonts
PFFDESCS  affdescsNames;    - Array of names
```

GpiQueryFontMetrics (Retrieves information about font metrics)

```
#define INCL_GPILCIDS

BOOL GpiQueryFontMetrics(hps, lMetricsLength, pfmMetrics);

HPS           hps;              - Presentation-space handle
LONG          lMetricsLength;   - Length of the structure
PFONTMETRICS  pfmMetrics;       - Address of structure for font metrics
```

GpiQueryFonts (Retrieves font information)

```
#define INCL_GPILCIDS

LONG GpiQueryFonts(hps, flOptions, pszFaceName, plRecFonts, lMetricsLength,
                   afmMetrics);

HPS           hps;              - Presentation-space handle
ULONG         flOptions;        - Type of fonts to retrieve
PSZ           pszFaceName;      - Address of typeface name of the fonts
PLONG         plRecFonts;       - Number of fonts to retrieve
LONG          lMetricsLength;   - Length of the structure
PFONTMETRICS  afmMetrics;       - Address of structures for font metrics
```

GpiQueryFullFontFileDescription (Retrieves family and face names of font file)

```
#define INCL_GPILCIDS

LONG GpiQueryFullFontFileDescriptions(hab, pszFileName, plCount, pNames,
                                      plNamesBuffLength);

HAB    hab;                 - Anchor-block handle
PSZ    pszFileName;         - Name of the font resource
PLONG  plCount;             - Number of pairs of descriptions returned
PVOID  pNames;              - Address of the buffer
PLONG  plNamesBuffLength;   - Address of doublewords in which buffer size
```

GpiQueryGraphicsField (Retrieves coordinates of a graphics field)

```
#define INCL_GPITRANSFORMS

BOOL GpiQueryGraphicsField(hps, prclField);

HPS     hps;          - Presentation-space handle
PRECTL  prclField;    - Address of structure for the graphics field
```

GpiQueryInitialSegmentAttrs (Retrieves an initial segment attribute)

```
#define INCL_GPISEGMENTS

LONG GpiQueryInitialSegmentAttrs(hps, lAttribute);

HPS   hps;           - Presentation-space handle
LONG  lAttribute;    - Attribute
```

GpiQueryKerningPairs (Retrieves kerning-pair information)

```
#define INCL_GPILCIDS

LONG GpiQueryKerningPairs(hps, lCount, akrnprData);

HPS            hps;          - Presentation-space handle
LONG           lCount;       - Number of kerning pairs
PKERNINGPAIRS  akrnprData;   - Address of array of kerning-pair structures
```

GpiQueryLineEnd (Retrieves the line-end attribute)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryLineEnd(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryLineJoin (Retrieves the line-join attribute)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryLineJoin(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryLineType (Retrieves the cosmetic line-type attribute)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryLineType(hps);

HPS  hps;     - Presentation-space handle
```

GpiQueryLineWidth (Retrieves the cosmetic line-width attribute)

```
#define INCL_GPIPRIMITIVES

FIXED GpiQueryLineWidth(hps);

HPS  hps;     - Presentation-space handle
```

GpiQueryLineWidthGeom (Retrieves the geometric line-width attribute)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryLineWidthGeom(hps);

HPS  hps;     - Presentation-space handle
```

GpiQueryLogColorTable (Retrieves the logical color table)

```
#define INCL_GPILOGCOLORTABLE

LONG GpiQueryLogColorTable(hps, flOptions, lStart, lCount, alArray);

HPS    hps;          - Presentation-space handle
ULONG  flOptions;    - Color type to retrieve
LONG   lStart;       - Starting index
LONG   lCount;       - Maximum number of values to copy
PLONG  alArray;      - Address of array for elements
```

GpiQueryLogicalFont (Retrieves description of a logical font)

```
#define INCL_GPILCIDS

BOOL GpiQueryLogicalFont(hps, lLcid, pName, pfatAttrs, lAttrsLength);

HPS      hps;             - Presentation-space handle
LONG     lLcid;           - Local identifier
PSTR8    pName;           - Address of logical-font name
PFATTRS  pfatAttrs;       - Address of structure for font attributes
LONG     lAttrsLength;    - Length of structure for font attributes
```

GpiQueryMarker (Retrieves the marker-symbol attribute)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryMarker(hps);

HPS  hps;     - Presentation-space handle
```

GpiQueryMarkerBox (Retrieves the marker-box attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryMarkerBox(hps, psizfxSize);

HPS     hps;           - Presentation-space handle
PSIZEF  psizfxSize;    - Address of structure for marker-box size
```

GpiQueryMarkerSet (Retrieves the marker-set attribute)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryMarkerSet(hps);

HPS  hps;     - Presentation-space handle
```

GpiQueryMetaFileBits (Transfers a metafile to application storage)

```
#define INCL_GPIMETAFILES

BOOL GpiQueryMetaFileBits(hmf, lOffset, lLength, pbData);

HMF    hmf;         - Metafile handle
LONG   lOffset;     - Offset to the first metafile byte to copy
LONG   lLength;     - Length in bytes of buffer
PBYTE  pbData;      - Address of buffer for metafile data
```

GpiQueryMetaFileLength (Retrieves the length of a memory metafile)

```
#define INCL_GPIMETAFILES

LONG GpiQueryMetaFileLength(hmf);

HMF  hmf;     - Metafile handle
```

GpiQueryMix (Retrieves the foreground mix mode)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryMix(hps);

HPS  hps;     - Presentation-space handle
```

GpiQueryModelTransformMatrix (Retrieves a model-transformation matrix)

```
#define INCL_GPITRANSFORMS

BOOL GpiQueryModelTransformMatrix(hps, lCount, pmatlfArray);

HPS        hps;           - Presentation-space handle
LONG       lCount;        - Number of elements
PMATRIXLF  pmatlfArray;   - Address of structure for transformation matrix
```

GpiQueryNearestColor (Retrieves the nearest available color)

```
#define INCL_GPILOGCOLORTABLE

LONG GpiQueryNearestColor(hps, flOptions, lRgbin);

HPS    hps;          - Presentation-space handle
ULONG  flOptions;    - Color type
LONG   lRgbin;       - RGB color value
```

GpiQueryNumberSetIds (Retrieves the number of lcids in use)

```
#define INCL_GPILCIDS

LONG GpiQueryNumberSetIds(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryPageViewport (Retrieves the coordinates of a page viewport)

```
#define INCL_GPITRANSFORMS

BOOL GpiQueryPageViewport(hps, prclViewport);

HPS     hps;             - Presentation-space handle
PRECTL  prclViewport;    - Address of structure for viewport
```

GpiQueryPalette (Retrieves a palette handle)

```
#define INCL_GPILOGCOLORTABLE

HPAL GpiQueryPalette(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryPaletteInfo (Retrieves information about a palette)

```
#define INCL_GPILOGCOLORTABLE

LONG GpiQueryPaletteInfo(hpal, hps, flOptions, ulStart, ulCount, aulArray);

HPAL   hpal;         - Palette handle
HPS    hps;          - Presentation-space handle
ULONG  flOptions;    - Options
ULONG  ulStart;      - Starting index for which data is returned
ULONG  ulCount;      - Count of elements
PLONG  aulArray;     - Address of array for palette information
```

GpiQueryPattern (Retrieves the shading-pattern attribute)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryPattern(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryPatternRefPoint (Retrieves the pattern reference point)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryPatternRefPoint(hps, pptlRefPoint);

HPS      hps;             - Presentation-space handle
PPOINTL  pptlRefPoint;    - Address of structure for pattern-reference point
```

GpiQueryPatternSet (Retrieves the pattern-set identifier)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryPatternSet(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryPel (Retrieves the color of a specified pel)

```
#define INCL_GPIPRIMITIVES

LONG GpiQueryPel(hps, pptlPoint);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlPoint;    - Address of structure with point to query
```

GpiQueryPickAperturePosition (Retrieves the center of a pick aperture)

```
#define INCL_GPICORRELATION

BOOL GpiQueryPickAperturePosition(hps, pptlPoint);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlPoint;    - Address of structure for center point
```

GpiQueryPickApertureSize (Retrieves the pick-aperture size)

```
#define INCL_GPICORRELATION

BOOL GpiQueryPickApertureSize(hps, psizlSize);

HPS     hps;          - Presentation-space handle
PSIZEL  psizlSize;    - Address of structure for pick-aperture size
```

GpiQueryPS (Retrieves presentation-space page parameters)

```
#define INCL_GPICONTROL

ULONG GpiQueryPS(hps, psizlSize);

HPS     hps;          - Presentation-space handle
PSIZEL  psizlSize;    - Address of structure for page size
```

GpiQueryRealColors (Retrieves RGB values)

```
#define INCL_GPILOGCOLORTABLE

LONG GpiQueryRealColors(hps, ulOptions, lStart, lCount, alColors);

HPS    hps;          - Presentation-space handle
ULONG  ulOptions;    - Color options
LONG   lStart;       - Ordinal number of first color
LONG   lCount;       - Number of colors
PLONG  alColors;     - Address of array of colors
```

GpiQueryRegionBox (Retrieves a region-box rectangle)

```
#define INCL_GPIREGIONS

LONG GpiQueryRegionBox(hps, hrgn, prclBound);

HPS     hps;          - Presentation-space handle
HRGN    hrgn;         - Region handle
PRECTL  prclBound;    - Address of structure for enclosing rectangle
```

GpiQueryRegionRects (Retrieves region rectangles)

```
#define INCL_GPIREGIONS

BOOL GpiQueryRegionRects(hps, hrgn, prclBound, prgnrcControl, arclRects);

HPS       hps;             - Presentation-space handle
HRGN      hrgn;            - Region handle
PRECTL    prclBound;       - Address of structure for enclosing rectangle
PRGNRECT  prgnrcControl;   - Address of structure controlling processing
PRECTL    arclRects;       - Address of array of rectangle structures
```

GpiQueryRGBColor (Retrieves an RGB color)

```
#define INCL_GPILOGCOLORTABLE

LONG GpiQueryRGBColor(hps, flOptions, lColorIndex);

HPS    hps;            - Presentation-space handle
ULONG  flOptions;      - Color type
LONG   lColorIndex;    - Color-index value
```

GpiQuerySegmentAttrs (Checks for a segment attribute)

```
#define INCL_GPISEGMENTS

LONG GpiQuerySegmentAttrs(hps, lSegid, lAttribute);

HPS   hps;           - Presentation-space handle
LONG  lSegid;        - Segment identifier
LONG  lAttribute;    - Attribute
```

GpiQuerySegmentNames (Retrieves the segment identifiers)

```
#define INCL_GPISEGMENTS

LONG GpiQuerySegmentNames(hps, lFirstSegment, lLastSegment, lMax,
                          palSegids);

HPS    hps;              - Presentation-space handle
LONG   lFirstSegment;    - First segment
LONG   lLastSegment;     - Last segment
LONG   lMax;             - Maximum number of segments
PLONG  palSegids;        - Address of array for segments
```

GpiQuerySegmentPriority (Retrieves the segment priority)

```
#define INCL_GPISEGMENTS

LONG GpiQuerySegmentPriority(hps, lRefSegid, lOrder);

HPS   hps;          - Presentation-space handle
LONG  lRefSegid;    - Reference-segment identifier
LONG  lOrder;       - Segment order
```

GpiQuerySegmentTransformMatrix (Retrieves segment-transformation matrix)

```
#define INCL_GPITRANSFORMS

BOOL GpiQuerySegmentTransformMatrix(hps, lSegid, lCount, pmatlfArray);

HPS         hps;            - Presentation-space handle
LONG        lSegid;         - Segment identifier
LONG        lCount;         - Number of elements
PMATRIXLF   pmatlfArray;    - Address of structure for matrix elements
```

GpiQuerySetIds (Retrieves information about fonts)

```
#define INCL_GPILCIDS

BOOL GpiQuerySetIds(hps, lCount, alTypes, aNames, allcids);

HPS    hps;         - Presentation-space handle
LONG   lCount;      - Number of objects to query
PLONG  alTypes;     - Address of array of types
PSTR8  aNames;      - Address of array for names
PLONG  alscids;     - Address of array for local identifiers
```

GpiQueryStopDraw (Retrieves the stop/draw condition)

```
#define INCL_GPICONTROL

LONG GpiQueryStopDraw(hps);

HPS  hps;    - Presentation-space handle
```

GpiQueryTag (Retrieves a tag identifier)

```
#define INCL_GPICORRELATION

BOOL GpiQueryTag(hps, plTag);

HPS    hps;       - Presentation-space handle
PLONG  plTag;     - Tag identifier
```

GpiQueryTextAlignment (Retrieves the text alignment attributes)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryTextAlignment(hps, plHorizontal, plVertical);

HPS    hps;             - Presentation-space handle
PLONG  plHorizontal;    - Address of horizontal alignment
PLONG  plVertical;      - Address of vertical alignment
```

GpiQueryTextBox (Retrieves the coordinates of a text box)

```
#define INCL_GPIPRIMITIVES

BOOL GpiQueryTextBox(hps, lCount1, pchString, lCount2, aptlPoints);

HPS      hps;           - Presentation-space handle
LONG     lCount1;       - Number of characters
PCH      pchString;     - Address of string
LONG     lCount2;       - Number of points
PPOINTL  aptlPoints;    - Address of array of point structures
```

GpiQueryViewingLimits (Retrieves coordinates of the viewing limits)

```
#define INCL_GPITRANSFORMS

BOOL GpiQueryViewingLimits(hps, prclLimits);

HPS     hps;           - Presentation-space handle
PRECTL  prclLimits;    - Address of structure for viewing limits
```

GpiQueryViewingTransformMatrix (Retrieves a viewing-transformation matrix)

```
#define INCL_GPITRANSFORMS

BOOL GpiQueryViewingTransformMatrix(hps, lCount, pmatlfArray);

HPS        hps;            - Presentation-space handle
LONG       lCount;         - Number of elements
PMATRIXLF  pmatlfArray;    - Address of structure for transformation matrix
```

GpiQueryWidthTable (Retrieves font-width-table information)

```
#define INCL_GPILCIDS

BOOL GpiQueryWidthTable(hps, lFirstChar, lCount, alData);

HPS    hps;            - Presentation-space handle
LONG   lFirstChar;     - Code point of first character
LONG   lCount;         - Number of elements
PLONG  alData;         - Address of array for width table
```

GpiRectInRegion (Determines whether rectangle is in a region)

```
#define INCL_GPIREGIONS

LONG GpiRectInRegion(hps, hrgn, prclRect);

HPS     hps;          - Presentation-space handle
HRGN    hrgn;         - Region handle
PRECTL  prclRect;     - Address of rectangle structure
```

GpiRectVisible (Determines whether a rectangle is visible)

```
#define INCL_GPIPRIMITIVES

LONG GpiRectVisible(hps, prclRect);

HPS     hps;          - Presentation-space handle
PRECTL  prclRect;     - Address of rectangle structure
```

GpiRemoveDynamics (Removes dynamic segments)

```
#define INCL_GPISEGMENTS

BOOL GpiRemoveDynamics(hps, lFirstSegid, lLastSegid)

HPS   hps;            - Presentation-space handle
LONG  lFirstSegid;    - First segment identifier
LONG  lLastSegid;     - Last segment identifier
```

GpiResetBoundaryData (Resets boundary data)

```
#define INCL_GPICORRELATION

BOOL GpiResetBoundaryData(hps);

HPS  hps;     - Presentation-space handle
```

GpiResetPS (Resets a presentation space)

```
#define INCL_GPICONTROL

BOOL GpiResetPS(hps, flOption);

HPS    hps;          - Presentation-space handle
ULONG  flOption;     - Reset option
```

GpiRestorePS (Restores a presentation space)

```
#define INCL_GPICONTROL

BOOL GpiRestorePS(hps, lPSid);

HPS   hps;       - Presentation-space handle
LONG  lPSid;     - Identifier for the presentation space
```

GpiRotate Applies rotation to a matrix transformation)

```
#define INCL_GPITRANSFORMS

BOOL GpiRotate(hps, pmatlfArray, lOptions, fxAngle, pptlCenter);

HPS        hps;            - Presentation-space handle
PMATRIXLF  pmatlfArray;    - Pointer to structure with matrix
LONG       lOptions;       - Transformation type
FIXED      fxAngle;        - Rotation angle
PPOINTL    pptlCenter;     - Pointer to structure with center point
```

GpiSaveMetaFile (Saves a metafile)

```
#define INCL_GPIMETAFILES

BOOL GpiSaveMetaFile(hmf, pszFileName);

HMF  hmf;            - Metafile handle
PSZ  pszFileName;    - Address of filename
```

GpiSavePS (Saves a presentation space)

```
#define INCL_GPICONTROL

LONG GpiSavePS(hps);

HPS  hps;    - Presentation-space handle
```

GpiScale (Scales a matrix-transformation operation)

```
#define INCL_GPITRANSFORMS

BOOL GpiScale(hps, pmatlfArray, lOptions, afxScale, pptlCenter);

HPS        hps;            - Presentation-space handle
PMATRIXLF  pmatlfArray;    - Pointer to structure for matrix
LONG       lOptions;       - Transformation options
PFIXED     afxScale;       - Pointer to variable with scaling factor
PPOINTL    pptlCenter;     - Pointer to structure with point data
```

GpiSelectPalette (Selects a palette into a presentation space)

```
#define INCL_GPILOGCOLORTABLE

HPAL GpiSelectPalette(hps, hpal);

HPS   hps;     - Presentation-space handle
HPAL  hpal;    - Palette handle
```

GpiSetArcParams (Sets the current arc parameters)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetArcParams(hps, parcpArcParams);

HPS         hps;              - Presentation-space handle
PARCPARAMS  parcpArcParams;   - Address of structure for arc parameters
```

GpiSetAttrMode (Sets the current attribute mode)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetAttrMode(hps, lMode);

HPS   hps;        - Presentation-space handle
LONG  lMode;    - Attribute mode
```

GpiSetAttrs (Sets the attributes for a primitive type)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetAttrs(hps, lPrimType, flAttrMask, flDefMask, pbunAttrs);

HPS      hps;           - Presentation-space handle
LONG     lPrimType;     - Primitive type
ULONG    flAttrMask;    - Attribute mask
ULONG    flDefMask;     - Defaults mask
PBUNDLE  pbunAttrs;     - Address of structure for attributes
```

GpiSetBackColor (Sets the current background color)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetBackColor(hps, lColor);

HPS   hps;       - Presentation-space handle
LONG  lColor;    - Background color
```

GpiSetBackMix (Sets the current background mix mode)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetBackMix(hps, lMixMode);

HPS   hps;         - Presentation-space handle
LONG  lMixMode;    - Background-mix mode
```

GpiSetBitmap (Sets a bit map)

```
#define INCL_GPIBITMAPS

HBITMAP GpiSetBitmap(hps, hbm);

HPS      hps;    - Presentation-space handle
HBITMAP  hbm;    - Bit-map handle
```

GpiSetBitmapBits (Sets the bits of a bit map)

```
#define INCL_GPIBITMAPS

LONG GpiSetBitmapBits(hps, lScanStart, lScans, pbBuffer, pbmi2InfoTable);

HPS           hps;              - Presentation-space handle
LONG          lScanStart;       - Index of first scan line
LONG          lScans;           - Number of scan lines
PBYTE         pbBuffer;         - Address of buffer with bit-map data
PBITMAPINFO2  pbmi2InfoTable;   - Address of structure with bit-map header table
```

GpiSetBitmapDimension (Sets the dimensions of a bit map)

```
#define INCL_GPIBITMAPS

BOOL GpiSetBitmapDimension(hbm, psizlBitmapDemension);

HBITMAP  hbm;                     - Bit-map handle
PSIZEL   psizlBitmapDemension;    - Address of structure with size of bit map
```

GpiSetBitmapId (Sets a bit-map identifier)

```
#define INCL_GPIBITMAPS

BOOL GpiSetBitmapId(hps, hbm, lLcid);

HPS      hps;       - Presentation-space handle
HBITMAP  hbm;       - Bit-map handle
LONG     lLcid;     - Local identifier
```

GpiSetCharAngle (Sets the character-angle attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetCharAngle(hps, pgradlAngle);

HPS         hps;            - Presentation-space handle
PGRADIENTL  pgradlAngle;    - Address of structure with baseline angle
```

GpiSetCharBox (Sets the character-box attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetCharBox(hps, psizfxBox);

HPS     hps;          - Presentation-space handle
PSIZEF  psizfxBox;    - Address of structure with character-box size
```

GpiSetCharBreakExtra (Sets the character-break-extra attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetCharBreakExtra(hps, fxBreakExtra);

HPS    hps;             - Presentation-space handle
FIXED  fxBreakExtra;    - Character-break-extra value
```

GpiSetCharDirection (Sets the character-direction attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetCharDirection(hps, lDirection);

HPS   hps;           - Presentation-space handle
LONG  lDirection;    - Character direction
```

GpiSetCharExtra (Sets the character-extra attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetCharExtra(hps, fxExtra);

HPS    hps;         - Presentation-space handle
FIXED  fxExtra;     - Character-extra value
```

GpiSetCharMode (Sets the character-mode attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetCharMode(hps, lMode);

HPS   hps;       - Presentation-space handle
LONG  lMode;     - Character mode
```

GpiSetCharSet (Sets the character-set identifier)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetCharSet(hps, lLcid);

HPS   hps;       - Presentation-space handle
LONG  lLcid;     - Local identifier
```

GpiSetCharShear (Sets the character-shear attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetCharShear(hps, pptlAngle);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlAngle;    - Address of structure with shear angle
```

GpiSetClipPath (Sets a clip path)

```
#define INCL_GPIPATHS

BOOL GpiSetClipPath(hps, lPath, lOptions);

HPS   hps;         - Presentation-space handle
LONG  lPath;       - Clip path identifier
LONG  lOptions;    - Options
```

GpiSetClipRegion (Sets a clip region)

```
#define INCL_GPIREGIONS

LONG GpiSetClipRegion(hps, hrgn, phrgnOld);

HPS    hps;          - Presentation-space handle
HRGN   hrgn;         - Region handle
PHRGN  phrgnOld;     - Address of variable for previous region handle
```

GpiSetColor (Sets the line-color attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetColor(hps, lColor);

HPS   hps;       - Presentation-space handle
LONG  lColor;    - Color value
```

GpiSetCp (Sets the graphics code-page identifier)

```
#define INCL_GPILCIDS

BOOL GpiSetCp(hps, ulCodePage);

HPS    hps;          - Presentation-space handle
ULONG  ulCodePage;   - Code-page identifier
```

GpiSetCurrentPosition (Sets the current position)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetCurrentPosition(hps, pptlPoint);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlPoint;    - Address of structure with new position
```

GpiSetDefArcParams (Sets the default arc parameters)

```
#define INCL_GPIDEFAULTS

BOOL GpiSetDefArcParams(hps, parcpArcParams);

HPS         hps;              - Presentation-space handle
PARCPARAMS  parcpArcParams;   - Pointer to structure with arc parameters
```

GpiSetDefAttrs (Sets the default attributes)

```
#define INCL_GPIDEFAULTS

BOOL GpiSetDefAttrs(hps, lPrimType, flAttrMask, pbunAttrs);

HPS      hps;          - Presentation-space handle
LONG     lPrimType;    - Primitive type
ULONG    flAttrMask;   - Attributes mask
PBUNDLE  pbunAttrs;    - Pointer to structure with default attributes
```

GpiSetDefaultViewMatrix (Sets the default viewing transformation)

```
#define INCL_GPITRANSFORMS

BOOL GpiSetDefaultViewMatrix(hps, lCount, pmatlfArray, lOptions);

HPS        hps;           - Presentation-space handle
LONG       lCount;        - Number of elements
PMATRIXLF  pmatlfArray;   - Address of structure with transform matrix
LONG       lOptions;      - Transformation type
```

GpiSetDefTag (Sets the default tag for a primitive)

```
#define INCL_GPIDEFAULTS

BOOL GpiSetDefTag(hps, lTag);

HPS   hps;     - Presentation-space handle
LONG  lTag;    - Tag
```

GpiSetDefViewingLimits (Sets the default viewing limits)

```
#define INCL_GPIDEFAULTS

BOOL GpiSetDefViewingLimits(hps, prclLimits);

HPS     hps;           - Presentation-space handle
PRECTL  prclLimits;    - Pointer to structure with viewing limits
```

GpiSetDrawControl (Sets a draw control)

```
#define INCL_GPICONTROL

BOOL GpiSetDrawControl(hps, lControl, lValue);

HPS   hps;          - Presentation-space handle
LONG  lControl;     - Draw control to change
LONG  lValue;       - Drawing flag
```

GpiSetDrawingMode (Sets the drawing mode)

```
#define INCL_GPICONTROL

BOOL GpiSetDrawingMode(hps, lMode);

HPS   hps;       - Presentation-space handle
LONG  lMode;     - Drawing mode
```

GpiSetEditMode (Sets the editing mode)

```
#define INCL_GPISEGEDITING

BOOL GpiSetEditMode(hps, lMode);

HPS   hps;       - Presentation-space handle
LONG  lMode;     - Editing mode
```

GpiSetElementPointer (Sets the element pointer)

```
#define INCL_GPISEGEDITING

BOOL GpiSetElementPointer(hps, lElement);

HPS   hps;          - Presentation-space handle
LONG  lElement;     - Element number
```

GpiSetElementPointerAtLabel (Sets the element pointer at a label)

```
#define INCL_GPISEGEDITING

BOOL GpiSetElementPointerAtLabel(hps, lLabel);

HPS   hps;        - Presentation-space handle
LONG  lLabel;     - Label identifier
```

GpiSetGraphicsField (Sets the coordinates of a graphics field)

```
#define INCL_GPITRANSFORMS

BOOL GpiSetGraphicsField(hps, prclField);

HPS     hps;          - Presentation-space handle
PRECTL  prclField;    - Address of structure with field
```

GpiSetInitialSegmentAttrs (Sets the initial segment attributes)

```
#define INCL_GPISEGMENTS

BOOL GpiSetInitialSegmentAttrs(hps, lAttribute, lAttrFlag);

HPS   hps;           - Presentation-space handle
LONG  lAttribute;    - Attribute type
LONG  lAttrFlag;     - Attribute on/lOffset flag
```

GpiSetLineEnd (Sets the line-end attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetLineEnd(hps, lLineEnd);

HPS   hps;          - Presentation-space handle
LONG  lLineEnd;     - Line end
```

GpiSetLineJoin (Sets the line-join attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetLineJoin(hps, lLineJoin);

HPS   hps;          - Presentation-space handle
LONG  lLineJoin;    - Line-join flags
```

GpiSetLineType (Sets the line-type attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetLineType(hps, lLineType);

HPS   hps;          - Presentation-space handle
LONG  lLineType;    - Line type
```

GpiSetLineWidth (Sets the cosmetic line-width attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetLineWidth(hps, fxLineWidth);

HPS    hps;           - Presentation-space handle
FIXED  fxLineWidth;   - Line width
```

GpiSetLineWidthGeom (Sets the geometric line-width attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetLineWidthGeom(hps, lLineWidth);

HPS   hps;          - Presentation-space handle
LONG  lLineWidth;   - Line width
```

GpiSetMarker (Sets the marker attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetMarker(hps, lSymbol);

HPS   hps;        - Presentation-space handle
LONG  lSymbol;    - Marker symbol
```

GpiSetMarkerBox (Sets the marker-box attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetMarkerBox(hps, psizfxSize);

HPS     hps;          - Presentation-space handle
PSIZEF  psizfxSize;   - Address of structure with marker box size
```

GpiSetMarkerSet (Sets the marker-set attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetMarkerSet(hps, lSet);

HPS   hps;     - Presentation-space handle
LONG  lSet;    - Local identifier
```

GpiSetMetaFileBits (Copies data from a buffer to a metafile)

```
#define INCL_GPIMETAFILES

BOOL GpiSetMetaFileBits(hmf, lOffset, lLength, pbBuffer);

HMF    hmf;          - Message-queue handle
LONG   lOffset;      - Offset into the metafile
LONG   lLength;      - Length of the metafile data
PBYTE  pbBuffer;     - Address of the metafile data
```

GpiSetMix (Sets the foreground mix mode)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetMix(hps, lMixMode);

HPS   hps;         - Presentation-space handle
LONG  lMixMode;    - Color-mixing mode
```

GpiSetModelTransformMatrix (Sets a model-transformation matrix)

```
#define INCL_GPITRANSFORMS

BOOL GpiSetModelTransformMatrix(hps, lCount, pmatlfArray, lOptions);

HPS        hps;            - Presentation-space handle
LONG       lCount;         - Number of elements
PMATRIXLF  pmatlfArray;    - Address of structure with transformation matrix
LONG       lOptions;       - Transformation types
```

GpiSetPageViewport (Sets the coordinates of a page viewport)

```
#define INCL_GPITRANSFORMS

BOOL GpiSetPageViewport(hps, prclViewport);

HPS     hps;             - Presentation-space handle
PRECTL  prclViewport;    - Address of structure with page viewport
```

GpiSetPaletteEntries (Changes the entries in a color palette)

```
#define INCL_GPILOGCOLORTABLE

BOOL GpiSetPaletteEntries (hpal, ulFormat, ulStart, ulCount, aTable);

HPAL    hpal;         - Palette handle
ULONG   ulFormat;     - Format of color-table entries
ULONG   ulStart;      - Starting index
ULONG   ulCount;      - Count of elements in color table
PULONG  aTable;       - Address of color table
```

GpiSetPattern (Sets the shading-pattern attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetPattern(hps, lPatternSymbol);

HPS   hps;               - Presentation-space handle
LONG  lPatternSymbol;    - Shading pattern
```

GpiSetPatternRefPoint (Sets the pattern reference point)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetPatternRefPoint(hps, pptlRefPoint);

HPS      hps;             - Presentation-space handle
PPOINTL  pptlRefPoint;    - Address of structure with reference point
```

GpiSetPatternSet (Sets the pattern-set attribute)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetPatternSet(hps, lSet);

HPS   hps;     - Presentation-space handle
LONG  lSet;    - Local identifier
```

GpiSetPel (Sets the color of a specified pel)

```
#define INCL_GPIBITMAPS

LONG GpiSetPel(hps, pptlPoint);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlPoint;    - Address of structure with point position
```

GpiSetPickAperturePosition (Sets the center of the pick aperture)

```
#define INCL_GPICORRELATION

BOOL GpiSetPickAperturePosition(hps, pptlPick);

HPS      hps;          - Presentation-space handle
PPOINTL  pptlPick;     - Address of structure with center of pick aperture
```

GpiSetPickApertureSize (Sets the pick-aperture size)

```
#define INCL_GPICORRELATION

BOOL GpiSetPickApertureSize(hps, lOptions, psizlSize);

HPS     hps;           - Presentation-space handle
LONG    lOptions;      - Options
PSIZEL  psizlSize;     - Address of structure with pick-aperture size
```

GpiSetPS (Sets presentation-space page parameters)

```
#define INCL_GPICONTROL

BOOL GpiSetPS(hps, psizlSize, flOptions);

HPS     hps;           - Presentation-space handle
PSIZEL  psizlSize;     - Address of structure for presentation-space size
ULONG   flOptions;     - Options
```

GpiSetRegion (Sets a region)

```
#define INCL_GPIREGIONS

BOOL GpiSetRegion(hps, hrgn, lCount, arclRectangles);

HPS     hps;               - Presentation-space handle
HRGN    hrgn;              - Region handle
LONG    lCount;            - Number of rectangles
PRECTL  arclRectangles;    - Address of array of rectangle structures
```

GpiSetSegmentAttrs (Sets an attribute for a retained segment)

```
#define INCL_GPISEGMENTS

BOOL GpiSetSegmentAttrs(hps, lSegid, lAttribute, lValue);

HPS   hps;           - Presentation-space handle
LONG  lSegid;        - Segment identifier
LONG  lAttribute;    - Attributes
LONG  lValue;        - Attribute on|lOffset flag
```

GpiSetSegmentPriority (Sets the segment priority)

```
#define INCL_GPISEGMENTS

BOOL GpiSetSegmentPriority(hps, lSegid, lRefSegid, lOrder);

HPS   hps;          - Presentation-space handle
LONG  lSegid;       - Segment identifier
LONG  lRefSegid;    - Reference-segment identifier
LONG  lOrder;       - Command options
```

GpiSetSegmentTransformMatrix (Sets a segment-transformation matrix)

```
#define INCL_GPITRANSFORMS

BOOL GpiSetSegmentTransformMatrix(hps, lSegid, lCount, pmatlfArray, lOptions);

HPS        hps;            - Presentation-space handle
LONG       lSegid;         - Segment identifier
LONG       lCount;         - Number of elements
PMATRIXLF  pmatlfArray;    - Address of structure with transformation matrix
LONG       lOptions;       - Type of transformation
```

GpiSetStopDraw (Sets the stop-draw condition)

```
#define INCL_GPICONTROL

BOOL GpiSetStopDraw(hps, lValue);

HPS   hps;       - Presentation-space handle
LONG  lValue;    - Stop-draw condition flag
```

GpiSetTag (Sets a tag for a primitive)

```
#define INCL_GPICORRELATION

BOOL GpiSetTag(hps, lTag);

HPS   hps;     - Presentation-space handle
LONG  lTag;    - Tag
```

GpiSetTextAlignment (Sets the alignment userd to postion of the text)

```
#define INCL_GPIPRIMITIVES

BOOL GpiSetTextAlignment(hps, lHorizontal, lVertical);

HPS   hps;           - Presentation-space handle
LONG  lHorizontal;   - Horizontal alignment
LONG  lVertical;     - Vertical alignment
```

GpiSetViewingLimits (Sets the coordinates of the viewing limits)

```
#define INCL_GPITRANSFORMS

BOOL GpiSetViewingLimits(hps, prclLimits);

HPS     hps;          - Presentation-space handle
PRECTL  prclLimits;   - Address of structure with viewing limits
```

GpiSetViewingTransformMatrix (Sets a viewing-transformation matrix)

```
#define INCL_GPITRANSFORMS

BOOL GpiSetViewingTransformMatrix(hps, lCount, pmatlfArray, lOptions);

HPS        hps;            - Presentation-space handle
LONG       lCount;         - Number of elements
PMATRIXLF  pmatlfArray;    - Address of structure with transformation matrix
LONG       lOptions;       - Transformation type
```

GpiStrokePath (Strokes a path)

```
#define INCL_GPIPATHS

LONG GpiStrokePath(hps, lPath, flOptions);

HPS    hps;          - Presentation-space handle
LONG   lPath;        - Identifier of stroke path
ULONG  flOptions;    - Reserved
```

GpiTranslate (Sets a matrix transformation)

```
#define INCL_GPITRANSFORMS

BOOL GpiTranslate(hps, pmatlfArray, lOptions, pptlTranslation);

HPS        hps;               - Presentation-space handle
PMATRIXLF  pmatlfArray;       - Pointer to structure with matrix
LONG       lOptions;          - Transformation type
PPOINTL    pptlTranslation;   - Pointer to structure with point data
```

GpiUnloadFonts (Unloads font definitions)

```
#define INCL_GPILCIDS

BOOL GpiUnloadFonts(hab, pszFileName);

HAB  hab;            - Anchor-block handle
PSZ  pszFileName;    - Address of the module name
```

GpiUnloadPublicFonts (Unloads fonts loaded by GpiLoadPublicFonts)

```
#define INCL_GPILCIDS

BOOL GpiUnloadPublicFonts(hab, pszFileName);

HAB  hab;            - Anchor-block handle
PSZ  pszFileName;    - Pointer to filename
```

GpiWCBitBlt (Copies a bit map to a presentation space)

```
#define INCL_GPIBITMAPS

LONG GpiWCBitBlt(hpsTarget, hbmSource, lCount, aptlPoints, lRop, flOptions);

HPS      hpsTarget;     - Presentation-space handle
HBITMAP  hbmSource;     - Bit-map handle
LONG     lCount;        - Number of points
PPOINTL  aptlPoints;    - Address of structure with points
LONG     lRop;          - Mixing function
ULONG    flOptions;     - Options
```

Prf Functions

PrfCloseProfile (Closes a profile file)

```
#define INCL_WINSHELLDATA

BOOL PrfCloseProfile(hini);

HINI  hini;     - Initialization-file handle
```

PrfOpenProfile (Opens a profile file)

```
#define INCL_WINSHELLDATA

HINI PrfOpenProfile(hab, pszFileName);

HAB  hab;            - Anchor-block handle
PSZ  pszFileName;    - Pointer to user-profile file name
```

PrfQueryProfile (Retrieves profile file names and descriptions)

```
#define INCL_WINSHELLDATA

BOOL PrfQueryProfile(hab, pprfproProfile);

HAB          hab;              - Anchor-block handle
PPRFPROFILE  pprfproProfile;   - Pointer to structure for profile data
```

PrfQueryProfileData (Retrieves information from the profile file)

```
#define INCL_WINSHELLDATA

BOOL PrfQueryProfileData(hini, pszApp, pszKey, pBuffer, pulBufferMax);

HINI    hini;           - Initialization-file handle
PSZ     pszApp;         - Pointer to application name
PSZ     pszKey;         - Pointer to keyname
PVOID   pBuffer;        - Pointer to buffer
PULONG  pulBufferMax;   - Buffer length
```

PrfQueryProfileInt (Retrieves an integer from the profile file)

```
#define INCL_WINSHELLDATA

LONG PrfQueryProfileInt(hini, pszAppName, pszKeyName, lDefault);

HINI  hini;       - Initialization-file handle
PSZ   pszApp;     - Pointer to application name
PSZ   pszKey;     - Pointer to keyname
LONG  lDefault;   - Default value returned if keyname not found
```

PrfQueryProfileSize (Retrieves size of keyname for program in profile)

```
#define INCL_WINSHELLDATA

BOOL PrfQueryProfileSize(hini, pszApp, pszKey, pDataLen);

HINI    hini;       - Initialization-file handle
PSZ     pszApp;     - Pointer to application name
PSZ     pszKey;     - Pointer to keyname
PULONG  pDataLen;   - Pointer to data length
```

PrfQueryProfileString (Retrieves a string from the profile file)

```
#define INCL_WINSHELLDATA

ULONG PrfQueryProfileString(hini, pszApp, pszKey, pszDefault, pBuffer,
                            cchBufferMax);

HINI   hini;            - Initialization-file handle
PSZ    pszApp;          - Pointer to application name
PSZ    pszKey;          - Pointer to keyname
PSZ    pszDefault;      - Pointer to default string
PVOID  pBuffer;         - Pointer to buffer for profile string
ULONG  cchBufferMax;    - Maximum length of buffer string
```

PrfReset (Resets Presentation Manager - defines files to be used as profiles)

```
#define INCL_WINSHELLDATA

BOOL PrfReset(hab, pprfpro);

HAB          hab;               - Anchor-block handle
PPRFPROFILE  pprfproProfile;    - Pointer to structure with profile data
```

PrfWriteProfileData (Places binary data in profile file)

```
#define INCL_WINSHELLDATA

BOOL PrfWriteProfileData(hini, pszApp, pszKey, pData, cchDataLen);

HINI   hini;          - Initialization-file handle
PSZ    pszApp;        - Pointer to application name
PSZ    pszKey;        - Pointer to keyname
PVOID  pData;         - Pointer to data in profile file
ULONG  cchDataLen;    - Data length
```

PrfWriteProfileString (Places a string in the profile file)

```
#define INCL_WINSHELLDATA

BOOL PrfWriteProfileString(hini, pszApp, pszKey, pszData);

HINI  hini;       - Initialization-file handle
PSZ   pszApp;     - Pointer to application name
PSZ   pszKey;     - Pointer to keyname
PSZ   pszData;    - Pointer to text string
```

Win Functions

WinAddAtom (Adds an atom to an atom table)

```
#define INCL_WINATOM

ATOM WinAddAtom(hatomtblAtomTbl, pszAtomName);

HATOMTBL  hatomtblAtomTbl;    - Handle to the atom table
PSZ       pszAtomName;        - Pointer to the atom name string
```

WinAddSwitchEntry (Adds an entry to the Window List)

```
#define INCL_WINSWITCHLIST

HSWITCH WinAddSwitchEntry(pswctlSwitchData);

PSWCNTRL  pswctlSwitchData;     - Address of structure with new entry information
```

WinAlarm (Generates an audible alarm)

```
#define INCL_WINDIALOGS

BOOL WinAlarm(hwndDeskTop, flStyle);

HWND   hwndDeskTop;     - Handle of the Desktop window
ULONG  flStyle;         - Alarm style
```

WinAssociateHelpInstance (Associates a help instance)

```
#define INCL_WINHELP

BOOL WinAssociateHelpInstance(hwndHelpInstance, hwndApp);

HWND  hwndHelpInstance;     - Handle of help instance
HWND  hwndApp;              - Application-window handle
```

WinBeginEnumWindows (Begins an enumeration process)

```
#define INCL_WINWINDOWMGR

HENUM WinBeginEnumWindows(hwndParent);

HWND  hwndParent;     - Handle of the parent window
```

WinBeginPaint (Obtains a presentation space ready for drawing)

```
#define INCL_WINWINDOWMGR

HPS WinBeginPaint(hwnd, hps, prclRect);

HWND    hwnd;         - Handle of the window
HPS     hps;          - Handle of the presentation space
PRECTL  prclRect;     - Bounding rectangle
```

WinBroadcastMsg (Broadcasts a message to multiple windows)

```
#define INCL_WINMESSAGEMGR

BOOL WinBroadcastMsg(hwndParent, ulMsgId, mpParam1, mpParam2, flCmd);

HWND    hwndParent;     - Handle of the parent window
ULONG   ulMsgId;        - Message indentifier
MPARAM  mpParam1;       - Message parameter 1
MPARAM  mpPAram2;       - Message parameter 2
ULONG   flCmd;          - Broadcast message command
```

WinCalcFrameRect (Calculates a client or frame rectangle)

```
#define INCL_WINFRAMEMGR

BOOL WinCalcFrameRect(hwnd, prclRect, fFrame);

HWND    hwnd;         - Handle of the frame window
PRECTL  prclRect;     - Window rectangle
BOOL    fFrame;       - Frame indicator
```

WinCallMsgFilter (Calls a message-filter hook)

```
#define INCL_WINHOOKS

BOOL WinCallMsgFilter(hab, pqmsgpqmsg, ulFilter);

HAB    hab;           - Handle of the anchor block
PQMSG  pqmsgpqmsg;    - Message passed to message-filter hook
ULONG  ulFilter;      - Message-filter
```

WinCancelShutdown (Cancels a request to shut down)

```
#define INCL_WINMESSAGEMGR

BOOL WinCancelShutdown(hmq, fCancelAlways);

HMQ   hmq;              - Handle of the message queue
BOOL  fCancelAlways;    - Cancellation control
```

WinChangeSwitchEntry (Changes information in a Window List entry)

```
#define INCL_WINSWITCHLIST

ULONG WinChangeSwitchEntry(hswitchSwitch, pswctlSwitchData);

HSWITCH   hswitchSwitch;       - Handle to Window List entry
PSWCNTRL  pswctlSwitchData;    - Switch control data
```

WinCheckButton (Sets the checked state of the button control)

```
#define INCL_WINWINDOWMGR

USHORT WinCheckButton (hwndDlg, usid, usChkstate);

HWND    hwndDlg;        - Dialog window handle
USHORT  usid;           - Buttom control identity
USHORT  usChkstate;     - Indicates the current checked state of button
```

WinCheckMenuItem (Sets the check state of the menu item)

```
#define INCL_WINWINDOWMGR

BOOL WinCheckMenuItem (hwndMenu, usid, fCheck);

HWND    hwndMenu;     - Menu window handle
USHORT  usid;         - Item identifier
BOOL    fCheck;       - Check flag
```

WinCloseClipbrd (Closes the clipboard)

```
#define INCL_WINCLIPBOARD

BOOL WinCloseClipbrd(hab);

HAB  hab;     - Handle of the anchor block
```

WinCompareStrings (Compares two strings)

```
#define INCL_WINCOUNTRY

ULONG WinCompareStrings(hab, idCodepage, idCountryCode, pszString1, pszString2,
                        flOptions);

HAB    hab;               - Handle of the anchor block
ULONG  idCodepage;        - Code-page identifier
ULONG  idCountryCode;     - Country code
PSZ    pszString1;        - Pointer to first string
PSZ    pszString2;        - Pointer to second string
ULONG  flOptions;         - Reserved; must be zero
```

WinCopyAccelTable (Retrieves accelerator-table data or size)

```
#define INCL_WINACCELERATORS

ULONG WinCopyAccelTable(hAccel, pacctAccelTable, ulCopyMax);

HACCEL       hAccel;             - Handle of the accelerator table
PACCELTABLE  pacctAccelTable;    - Accelerator-table data area
ULONG        ulCopyMax;          - Maximum size of data area
```

WinCopyRect (Copies a rectangle)

```
#define INCL_WINRECTANGLES

BOOL WinCopyRect(hab, prclDest, prclSrc);

HAB     hab;          - Handle of the anchor block
PRECTL  prclDest;     - Pointer to destination rectangle
PRECTL  prclSrc;      - Pointer to source rectangle
```

WinCpTranslateChar (Translates a character between code pages)

```
#define INCL_WINCOUNTRY

UCHAR WinCpTranslateChar(hab, ulCpSource, ucSource, ulCpDest);

HAB    hab;           - Handle of the anchor block
ULONG  ulCpSource;    - Code page of source character
UCHAR  ucSource;      - Character to be translated
ULONG  ulCpDest;      - Code page of resultant character
```

WinCpTranslateString (Translates a string between code pages)

```
#define INCL_WINCOUNTRY

BOOL WinCpTranslateString(hab, ulCpSource, pszSource, ulCpDest, ulLenDest, pszDest);

HAB    hab;          - Handle of the anchor block
ULONG  ulCpSource;   - Code page of source string
PSZ    pszSource;    - Pointer to string to be translated
ULONG  ulCpDest;     - Code page of resultant string
ULONG  ulLenDest;    - Maximum length of output string
PSZ    pszDest;      - Pointer to translated string
```

WinCreateAccelTable (Creates an accelerator table)

```
#define INCL_WINACCELERATORS

HACCEL WinCreateAccelTable(hab, pacctAccelTable);

HAB          hab;              - Handle of the anchor block
PACCELTABLE  pacctAccelTable;  - Pointer to accelerator table
```

WinCreateAtomTable (Creates an empty atom table)

```
#define INCL_WINATOM

HATOMTBL WinCreateAtomTable(ulInitial, ulBuckets);

ULONG  ulInitial;    - Initial bytes for atom table
ULONG  ulBuckets;    - Size of hash table
```

WinCreateCursor (Creates a cursor for a specified window)

```
#define INCL_WINCURSORS

BOOL WinCreateCursor(hwnd, lx, ly, lcx, lcy, ulrgf, prclClip);

HWND    hwnd;        - Handle of the window with cursor
LONG    lx;          - Horizontal position of the cursor
LONG    ly;          - Vertical position of the cursor
LONG    lcx;         - Cursor width
LONG    lcy;         - Cursor height
ULONG   ulrgf;       - Cursor appearance
PRECTL  prclClip;    - Pointer to cursor rectangle
```

WinCreateDlg (Creates a dialog window)

```
#define INCL_WINDIALOGS

HWND WinCreateDlg(hwndParent, hwndOwner, pDlgProc, pdlgtDlgTmp, pCreateParams);

HWND          hwndParent;       - Handle of the parent window
HWND          hwndOwner;        - Handle of the owner window
PFNWP         pDlgProc;         - Pointer to dialog procedure
PDLGTEMPLATE  pdlgtDlgTmp;      - Pointer to dialog template
PVOID         pCreateParams;    - Pointer to application-defined data area
```

WinCreateFrameControls (Creates the standard frame controls)

```
#define INCL_WINFRAMEMGR

BOOL WinCreateFrameControls(hwndFrame, pFcdata, pszTitle);

HWND         hwndFrame;     - Handle of the frame window
PFRAMECDATA  pFcdata;       - Frame-control data
PSZ          pszTitle;      - Pointer to title string
```

WinCreateHelpInstance (Creates a help instance)

```
#define INCL_WINHELP

HWND WinCreateHelpInstance(hab, phinitHMInitStructure);

HAB        hab;                        - Anchor-block handle
PHELPINIT  phinitHMInitStructure;      - Pointer to help initialization structure
```

WinCreateHelpTable (Identifies or changes a help table)

```
#define INCL_WINHELP

BOOL WinCreateHelpTable(hwndHelpInstance, phtHelpTable);

HWND        hwndHelpInstance;     - Handle of help instance
PHELPTABLE  phtHelpTable;         - Pointer to help table
```

WinCreateMenu (Creates a menu window)

```
#define INCL_WINMENUS

HWND WinCreateMenu(hwndOwner, pmtMenutmp);

HWND   hwndOwner;      - Handle of the owner/parent window
PVOID  pmtMenutmp;     - Pointer to the menu template
```

WinCreateMsgQueue (Creates a message queue)

```
#define INCL_WINMESSAGEMGR

HMQ WinCreateMsgQueue(hab, lQueuesize);

HAB   hab;           - Handle of the anchor block
LONG  lQueuesize;    - Maximum size of the message queue
```

WinCreateObject (Creates an instance of object class)

```
#define INCL_WINWORKPLACE

HOBJECT WinCreateObject(pszClassName, pszTitle, pszSetupString, pszLocation,
                        ulFlags);

PSZ    pszClassName;       - Pointer to string containing name of class
PSZ    pszTitle;           - Pointer to string containing initial title of object
PSZ    pszSetupString;     - Pointer to setup string
PSZ    pszLocation;        - Pointer to folder location
ULONG  ulFlags;            - Creation flags
```

WinCreatePointer (Creates a pointer from a bit map)

```
#define INCL_WINPOINTERS

HPOINTER WinCreatePointer(hwndDeskTop, hbmBitMap, fPointerSize, lxHotspot,
                          lyHotspot);

HWND     hwndDeskTop;      - Handle of the Desktop window
HBITMAP  hbmBitMap;        - Handle of the bit map
BOOL     fPointerSize;     - Full-size or icon-size pointer
LONG     xHotspot;         - Horizontal hot spot
LONG     yHotspot;         - Vertical hot spot
```

WinCreatePointerIndirect (Creates a color pointer from a bit map)

```
#define INCL_WINPOINTERS

HPOINTER WinCreatePointerIndirect(hwndDeskTop, pptriPointerInfo);

HWND          hwndDeskTop;        - Desktop handle
PPOINTERINFO  pptriPoinerInfo;    - Pointer to structure with bit map
```

WinCreateStdWindow (Creates a standard frame window)

```
#define INCL_WINFRAMEMGR

HWND WinCreateStdWindow(hwndParent, flStyle, pflCreateFlags, pszClientClass,
                        pszTitle, flClientStyle, Resource, ulId, phwndClient);

HWND     hwndParent;         - Handle of the parent window
ULONG    flStyle;            - Frame-window style
PULONG   pflCreateFlags;     - Creation flags
PSZ      pszClientClass;     - Client-window class name
PSZ      pszTitle;           - Address of title-bar text
ULONG    flClientStyle;      - Client-window style
HMODULE  Resource;           - Resouce identifier
ULONG    ulId;               - Frame-window identifier
PHWND    phwndClient;        - Address of client-window handle
```

WinCreateSwitchEntry (Creates an entry in the Window List)

```
#define INCL_WINSWITCHLIST

HSWITCH WinCreateSwitchEntry(hab, pswctlSwitchData);

HAB       hab;                - Anchor-block handle
PSWCNTRL  pswctlSwitchData;   - Pointer to structure with new entry information
```

WinCreateWindow (Creates a window)

```
#define INCL_WINWINDOWMGR

HWND WinCreateWindow(hwndParent, pszClass, pszName, flStyle, x, y, cx, cy,
                     hwndOwner, hwndInsertBehind, Id, pCtlData, pPresParams);

HWND   hwndParent;          - Parent-window handle
PSZ    pszClass;            - Pointer to registered class name
PSZ    pszName;             - Pointer to window text
ULONG  flStyle;             - Window style
LONG   x;                   - Horizontal position of window
LONG   y;                   - Vertical position of window
LONG   cx;                  - Window width
LONG   cy;                  - Window height
HWND   hwndOwner;           - Owner-window handle
HWND   hwndInsertBehind;    - Handle to sibling window
ULONG  Id;                  - Window identifier
PVOID  pCtlData;            - Pointer to buffer
PVOID  pPresParams;         - Pointer to structure with pres. params
```

WinDdeInitiate (Sends a WM_DDE_INITIATE message to all windows)

```
#define INCL_WINDDE

BOOL WinDdeInitiate2(hwndClient, pszAppName, pszTopicName, pContext);

HWND          hwndClient;      - Handle of the client window
PSZ           pszAppName;      - Address of application name
PSZ           pszTopicName;    - Address of topic name
PCONVCONTEXT  pContext;        - Address of CONVCONTEXT structure
```

WinDdePostMsg (Posts a message to a message queue)

```
#define INCL_WINDDE

BOOL WinDdePostMsg2(hwndTo, hwndFrom, usMsgId, pData, ulOptions);

HWND        hwndTo;       - Handle of the window to post to
HWND        hwndFrom;     - Handle of the window that is posting
USHORT      usMsgId;      - Message number
PDDESTRUCT  pData;        - Address of the structure to pass
ULONG       flOptions;    - Options
```

WinDdeRespond (Responds to sender of a WM_DDE_INITIATE message)

```
#define INCL_WINDDE

MRESULT WinDdeRespond2(hwndClient, hwndServer, pszAppName, pszTopicName, pContext);

HWND          hwndClient;      - Handle of the client window
HWND          hwndServer;      - Handle of the server window
PSZ           pszAppName;      - Address of name of application
PSZ           pszTopicName;    - Address of name of topic
PCONVCONTEXT  pContext;        - Address of CONVCONTEXT structure
```

WinDefDlgProc (Calls the default dialog procedure)

```
#define INCL_WINDIALOGS

MRESULT WinDefDlgProc(hwndDlg, ulMsgid, mpParam1, mpParam2);

HWND    hwndDlg;      - Handle of the dialog
ULONG   ulMsgid;      - Message identity
MPARAM  mpParam1;     - Message parameter
MPARAM  mpParam2;     - Message parameter
```

WinDefFileDlgProc (Calls the default file dialog procedure)

```
#define INCL_WINSTDFILE

MRESULT WinDefFileDlfProc(hwndDlg, ulMsgid, mpParam1, mpParam2);

HWND    hwndDlg;      - Handle of the dialog
ULONG   ulMsgid;      - Message identity
MPARAM  mpParam1;     - Message parameter
MPARAM  mpParam2;     - Message parameter
```

WinDefFontDlgProc (Calls the default font fialog procedure)

```
#define INCL_WINSTDFONT

MRESULT WinDefFontDlfProc(hwndDlg, ulMsgid, mp1, mpParam2);

HWND    hwndDlg;      - Handle of the dialog
ULONG   ulMsgid;      - Message idenity
MPARAM  mpParam1;     - Message parameter
MPARAM  mpParam2;     - Message parameter
```

WinDefWindowProc (Calls the default window procedure)

```
#define INCL_WINWINDOWMGR

MRESULT WinDefWindowProc(hwnd, ulMsgid, mpParam1, mpParam2);

HWND    hwnd;         - Handle of the window
ULONG   ulMsgid;      - Message identity
MPARAM  mpParam1;     - Message parameter
MPARAM  mpParam2;     - Message parameter
```

WinDeleteAtom (Deletes an atom from an atom table)

```
#define INCL_WINATOM

ATOM WinDeleteAtom(hatomtblAtomTbl, atom);

HATOMTBL  hatomtblAtomTbl;     - Handle of the atom table
ATOM      atom;                - Atom
```

WinDeleteLboxItem (Deletes the indexed item from the listbox)

```
#define INCL_WINWINDOWMGR

SHORT WinDeleteLboxItem(hwndLbox, sIndex);

HWND   hwndLbox;     - Listbox handle
SHORT  sIndex;       - Index of the listbox item
```

WinDeleteLibrary (Deletes a dynamic-link library)

```
#define INCL_WINLOAD

BOOL WinDeleteLibrary(hab, hlibLibhandle);

HAB   hab;              - Anchor-block handle
HLIB  hlibLibhandle;    - Handle of library to be deleted
```

WinDeleteProcedure (Deletes a procedure)

```
#define INCL_WINLOAD

BOOL WinDeleteProcedure(hab, pwndProc);

HAB    hab;         - Anchor-block handle
PFNWP  pwndProc;    - Pointer to window function
```

WinDeregisterObjectClass (Removes a workplace object class)

```
#define INCL_WINWORKPLACE

BOOL WinDeregisterObjectClass(pszClassName);

PSZ  pszClassName;    - Pointer to the name of the object
```

WinDestroyAccelTable (Destroys an accelerator table)

```
#define INCL_WINACCELERATORS

BOOL WinDestroyAccelTable(haccelAccel);

HACCEL  haccelAccel;    - Handle of the accelerator table
```

WinDestroyAtomTable (Destroys an atom table)

```
#define INCL_WINATOM

HATOMTBL WinDestroyAtomTable(hatomtblAtomTbl);

HATOMTBL  hatomtblAtomTbl;    - Handle of the atom table
```

WinDestroyCursor (Destroys the current cursor)

```
#define INCL_WINCURSORS

BOOL WinDestroyCursor(hwnd);

HWND  hwnd;    - Handle of the window
```

WinDestroyHelpInstance (Destroys a help instance)

```
#define INCL_WINHELP

BOOL WinDestroyHelpInstance(hwndHelpInstance);

HWND  hwndHelpInstance;    - Handle of instance to destroy
```

WinDestroyMsgQueue (Destroys a message queue)

```
#define INCL_WINMESSAGEMGR

BOOL WinDestroyMsgQueue(hmq);

HMQ  hmq;     - Handle of the message queue
```

WinDestroyObject (Delete a workplace object)

```
#define INCL_WINWORKPLACE

BOOL WinDestroyObject(object);

HOBJECT  object;     - Handle to a workplace object
```

WinDestroyPointer (Destroys a pointer or icon)

```
#define INCL_WINPOINTERS

BOOL WinDestroyPointer(hptrPointer);

HPOINTER  hptrPointer;     - Handle of the pointer or icon
```

WinDestroyWindow (Destroys a window)

```
#define INCL_WINWINDOWMGR

BOOL WinDestroyWindow(hwnd);

HWND  hwnd;     - Handle of the window to destroy
```

WinDismissDlg (Hides a dialog window)

```
#define INCL_WINDIALOGS

BOOL WinDismissDlg(hwndDlg, usResult);

HWND   hwndDlg;      - Handle of the dialog
ULONG  usResult;     - Result code to return
```

WinDispatchMsg (Dispatches a message)

```
#define INCL_WINMESSAGEMGR

MRESULT WinDispatchMsg(hab, pqmsgMsg);

HAB    hab;          - Handle of the anchor block
PQMSG  pqmsgMsg;     - Address of structure for message queue
```

WinDlgBox (Loads and processes a modal dialog box)

```
#define INCL_WINDIALOGS

ULONG WinDlgBox(hwndParent, hwndOwner, pfnDlgProc, hmod, idDlg, pCreateParams);

HWND     hwndParent;       - Handle of the parent window
HWND     hwndOwner;        - Handle of the owner window
PFNWP    pfnDlgProc;       - Dialog procedure address
HMODULE  hmod;             - Handle of resource module
ULONG    idDlg;            - Identifies the dialog
PVOID    pCreateParams;    - Address of application-specific data
```

WinDrawBitmap (Draws a bit map)

```
#define INCL_WINWINDOWMGR

BOOL WinDrawBitmap(hps, hbm, prclSrc, pptlDest, lForeColor, lBackColor, flRgf);

HPS      hps;          - Handle of the destination presentation space
HBITMAP  hbm;          - Handle of the bit map
PRECTL   prclSrc;      - Address of structure with rectangle coordinates
PPOINTL  pptlDest;     - Address of structure with bit-map position
LONG     lForeColor;   - Color of the foreground queuing of physical input
LONG     lBackColor;   - Color of the backgroundbled state
ULONG    flRgf;        - Bit-map-drawing flagspdating
```

WinDrawBorder (Draws a border)

```
#define INCL_WINWINDOWMGR

BOOL WinDrawBorder(hps, prclRect, lVertSideWidth, lHorizSideWidth, lBorderColor,
                   lInteriorColor, flCmd);

HPS     hps;               - Handle of the presentation space
PRECTL  prclRect;          - Address of structure with bounding rectangle
LONG    lVertSideWidth;    - Width of the border
LONG    lHorizSideWidth;   - Height of the border
LONG    lBorderColor;      - Color of the foreground
LONG    lInteriorColor;    - Color of the background
ULONG   flCmd;             - Border-drawing flags
```

WinDrawPointer (Draws a pointer)

```
#define INCL_WINPOINTERS

BOOL WinDrawPointer(hps, lx, ly, hptrPointer, ulHalfTone);

HPS       hps;           - Handle of the presentation space
LONG      lx;            - Horizontal position
LONG      ly;            - Vertical position
HPOINTER  hptrPointer;   - Handle to the mouse pointer
ULONG     ulHalfTone;    - Pointer-drawing flags
```

WinDrawText (Draws a single line of formatted text)

```
#define INCL_WINWINDOWMGR

SHORT WinDrawText(hps, lCount, pchText, pcrlRect, lForeColor, lBackColor, flCmd);

HPS     hps;           - Handle of the presentation space
LONG    lCount;        - Number of characters in the string
PCH     pchText;       - Address of the text
PRECTL  prclRect;      - Address of structure with formatting dimensions
LONG    lForeColor;    - Color of the foreground
LONG    lBackColor;    - Color of the background
ULONG   flCmd;         - Text-drawing flags
```

WinEmptyClipbrd (Empties the clipboard)

```
#define INCL_WINCLIPBOARD

BOOL WinEmptyClipbrd(hab);

HAB  hab;     - Handle of the anchor block
```

WinEnableControl (Sets the enable state of the item in dialog)

```
#define INCL_WINWINDOWMGR

BOOL WinEnableControl(hwndDlg, usId, fEnable);

HWND    hwndDlg;     - Dialog window handle
USHORT  usId;        - Identity of the item in the dialog template
BOOL    fEnable;     - Enable flag
```

WinEnableMenuItem (Sets the enable etate of the item in the menu)

```
#define INCL_WINWINDOWMGR

BOOL WinEnableMenuItem(hwndMenu, usId, fEnable);

HWND    hwndMenu;     - Menu window handle
USHORT  usId;         - Item identifier
BOOL    fEnable;      - Enable flag
```

WinEnablePhysInput (Enables or disables queuing of physical input)

```
#define INCL_WININPUT

BOOL WinEnablePhysInput(hwndDeskTop, fNewInputState);

HWND  hwndDeskTop;       - Handle of the Desktop
BOOL  fNewInputState;    - Input status
```

WinEnableWindow (Sets the window-enabled state)

```
#define INCL_WINWINDOWMGR

BOOL WinEnableWindow(hwnd, fNewEnabled);

HWND  hwnd;           - Handle of the window
BOOL  fNewEnabled;    - Enable-state flag
```

WinEnableWindowUpdate (Enables window updating)

```
#define INCL_WINWINDOWMGR

BOOL WinEnableWindowUpdate(hwnd, fNewVisibility);

HWND  hwnd;              - Handle of the window to be enabled or disabled
BOOL  fNewVisibility;    - Enabled-state flag
```

WinEndEnumWindows (Ends the enumeration process)

```
#define INCL_WINWINDOWMGR

BOOL WinEndEnumWindows(henum);

HENUM  henum;     - Handle of the enumeration list
```

WinEndPaint (Indicates that redrawing of a window is complete)

```
#define INCL_WINWINDOWMGR

BOOL WinEndPaint(hps);

HPS  hps;     - Handle of the presentation space
```

WinEnumClipbrdFmts (Enumerates the list of clipboard data formats)

```
#define INCL_WINCLIPBOARD

ULONG WinEnumClipbrdFmts(hab, fmt);

HAB    hab;     - Handle of the anchor block
ULONG  fmt;     - Index of last format enumerated
```

WinEnumDlgItem (Returns the window handle of a dialog item)

```
#define INCL_WINDIALOGS

HWND WinEnumDlgItem(hwndDlg, hwnd, ulCode);

HWND   hwndDlg;     - Handle of the dialog window
HWND   hwnd;        - Handle of the child window
ULONG  ulCode;      - Dialog item to return
```

WinEnumObjectClass (Returned a list of all worlplace object classes)

```
#define INCL_WINWORKPLACE

BOOL WinEnumObjectClass(pObjClass, pSize);

POBJCLASS  pObjClass;     - Pointer to a buffer
PULONG     pSize;         - Pointer to a buffer size
```

WinEqualRect (Compares two rectangles for equality)

```
#define INCL_WINRECTANGLES

BOOL WinEqualRect(hab, prclRect1, prclRect2);

HAB     hab;          - Handle of the anchor block
PRECTL  prclRect1;    - Address of structure for first rectangle
PRECTL  prclRect2;    - Address of structure for second rectangle
```

WinExcludeUpdateRegion (Excludes an update region)

```
#define INCL_WINWINDOWMGR

LONG WinExcludeUpdateRegion(hps, hwnd);

HPS   hps;      - Handle of the presentation space
HWND  hwnd;     - Handle of the window
```

WinFileDlg (Create and displays the file dialog)

```
#define INCL_WINSTDFILE

HWND WinFileDlg(hwndP, hwndO, pfiId);

HWND      hwndP;     - Parent-window handle
HWND      hwndO;     - Owner-window handle
PFILEDLG  pfiId;     - Pointer to FILEDLG
```

WinFillRect (Draws a filled rectangular area)

```
#define INCL_WINWINDOWMGR

BOOL WinFillRect(hps, pcrlRect, lColor);

HPS     hps;          - Handle of the presentation space
PRECTL  pcrlRect;     - Address of the structure with the rectangle
LONG    lColor;       - Color of the rectangle
```

WinFindAtom (Finds an atom in the atom table)

```
#define INCL_WINATOM

ATOM WinFindAtom(hatomtblAtomTbl, pszAtomName);

HATOMTBL  hatomtblAtomTbl;     - Handle of the atom table
PSZ       pszAtomName;         - Address of the atom name
```

WinFlashWindow (Starts or stops flashing a window)

```
#define INCL_WINFRAMEMGR

BOOL WinFlashWindow(hwndFrame, fFlash);

HWND  hwndFrame;     - Handle of the window to flash
BOOL  fFlash;        - Start/stop flashing flag
```

WinFocusChange (Changes the focus)

```
#define INCL_WININPUT

BOOL WinFocusChange(hwndDeskTop, hwndNewFocus, flFocusChange);

HWND   hwndDeskTop;       - Handle of the Desktop
HWND   hwndNewFocus;      - Handle of the focus window
ULONG  flFocusChange;     - Focus-changing flags
```

WinFontDlg (Create and displays the font dialog)

```
#define INCL_WINSTDFONT

HWND WinFontDlg(hwndP, hwndO, pfntd);

HWND      hwndP;     - Parent-window handle
HWND      hwndO;     - Owner-window handle
PFONTDLG  pfntd;     - Pointer to FONTDLG
```

WinFreeErrorInfo (Frees memory allocated for error information)

```
#define INCL_WINSTDFILE

BOOL WinFreeErrorInfo(perriErrorInfo);

PERRINFO  perriErrorInfo;     - Address of structure with error-info block
```

WinFreeFileDlgList (Frees memory allocated for file dialog)

```
#define INCL_WINSTDFILE

BOOL WinFreeFileDlgList(papszFQFilename);

PAPSZ  papszFQFilenmae;     - Pointer to a table of pointer to list
```

WinFreeFileIcon (Frees the pointer an icon)

```
#define INCL_WINWORKPLACE

BOOL WinFreeFileIcon(hptr);

HPOINTER  hptr;     - Pointer to a icon loaded by WinLoadFileIcon
```

WinGetClipPS (Retrieves a clipped presentation space)

```
#define INCL_WINWINDOWMGR

HPS WinGetClipPS(hwnd, hwndClipWindow, ulClipFlags);

HWND   hwnd;              - Address of the parent window
HWND   hwndClipWindow;    - Handle of clipping type
ULONG  ulClipFlags;       - Clipping flags
```

WinGetCurrentTime (Retrieves the current time)

```
#define INCL_WINTIMER

ULONG WinGetCurrentTime(hab);

HAB  hab;     - Handle of the anchor block
```

WinGetDlgMsg (Retrieves a dialog message)

```
#define INCL_WINDIALOGS

BOOL WinGetDlgMsg(hwndDlg, pqmsgmsg);

HWND   hwndDlg;      - Dialog-window handle
PQMSG  pqmsgmsg;     - Pointer to structure with message
```

WinGetErrorInfo (Retrieves the error information)

```
#define INCL_WINERRORS

PERRINFO WinGetErrorInfo(hab);

HAB  hab;     - Handle of the anchor block
```

WinGetKeyState (Retrieves the key state)

```
#define INCL_WININPUT

LONG WinGetKeyState(hwndDeskTop, vkey);

HWND  hwndDeskTop;     - Handle of the Desktop
LONG  vkey;            - Virtual key
```

WinGetLastError (Retrieves the last error state)

```
#define INCL_WINERRORS

ERRORID WinGetLastError(hab);

HAB  hab;     - Handle of the anchor block
```

WinGetMaxPosition (Retrieves the maximized-window position)

```
#define INCL_WINFRAMEMGR

BOOL WinGetMaxPosition(hwnd, pSwp);

HWND  hwnd;     - Handle of the window
PSWP  pSwp;     - Address of structure for maximum window size and position
```

WinGetMinPosition (Retrieves the minimized-window position)

```
#define INCL_WINFRAMEMGR

BOOL WinGetMinPosition(hwnd, pSwp, pptlPoint);

HWND     hwnd;          - Handle of the window
PSWP     pSwp;          - Address of structure with icon information
PPOINTL  pptlPoint;     - Address of structure with minimum window position
```

WinGetMsg (Retrieves a message from a message queue)

```
#define INCL_WINMESSAGEMGR

BOOL WinGetMsg(hab, pqmsgmsg, hwndFilter, ulFirst, ulLast);

HAB    hab;           - Handle of the anchor block
PQMSG  pqmsgmsg;      - Address of structure with message
HWND   hwndFilter;    - Window-filter handle
ULONG  ulFirst;       - First message
ULONG  ulLast;        - Last message
```

WinGetNextWindow (Retrieves next window from enumeration list)

```
#define INCL_WINWINDOWMGR

HWND WinGetNextWindow(henum);

HENUM  henum;     - Handle of the enumeration list
```

WinGetPhysKeyState (Retrieves the physical key state)

```
#define INCL_WININPUT

LONG WinGetPhysKeyState(hwndDeskTop, sc);

HWND  hwndDeskTop;     - Handle of the Desktop
LONG  sc;              - Scan code of the key
```

WinGetPS (Retrieves a cached presentation space)

```
#define INCL_WINWINDOWMGR

HPS WinGetPS(hwnd);

HWND  hwnd;     - Handle of the window
```

WinGetScreenPS (Retrieves a screen presentation space)

```
#define INCL_WINWINDOWMGR

HPS WinGetScreenPS(hwndDeskTop);

HWND  hwndDeskTop;     - Handle of the Desktop
```

WinGetSysBitmap (Retrieves a handle to a standard system bit map)

```
#define INCL_WINPOINTERS

HBITMAP WinGetSysBitmap(hwndDeskTop, ulIndex);

HWND   hwndDeskTop;     - Handle of the Desktop
ULONG  ulIndex;         - Index of the system bit map
```

WinInflateRect (Expands a rectangle)

```
#define INCL_WINRECTANGLES

BOOL WinInflateRect(hab, prclRect, lcx, lcy);

HAB     hab;          - Handle of the anchor block
PRECTL  prclRect;     - Address of structure with expanded rectangle
LONG    lcx;          - Amount to expand width
LONG    lcy;          - Amount to expand height
```

WinInitialize (Initializes thread for Presentation Manager calls)

```
#define INCL_WINWINDOWMGR

HAB WinInitialize(flOptions);

ULONG  flOptions;     - Initialization options
```

WinInSendMsg (Determines whether thread is processing a message)

```
#define INCL_WINMESSAGEMGR

BOOL WinInSendMsg(hab);

HAB hab;     - Handle of the anchor block
```

WinInsertLbpxItem (Inserts text into a listbox at index)

```
#define INCL_WINWINDOWMGR

SHORT WinInsertLboxItem(hwndLbox, sIndex, pszText);

HWND   hwndLbox;     - List box handle
SHORT  sIndex;       - Idex of the list box item
PSZ    pszText;      - Text to be inserted
```

WinIntersectRect (Calculates intersection of two source rectangles)

```
#define INCL_WINRECTANGLES

BOOL WinIntersectRect(hab, prclDest, prclRect1, prclRect2);

HAB     hab;          - Handle of the anchor block
PRECTL  prclDest;     - Address of structure for intersection of rectangles
PRECTL  prclRect1;    - Address of structure with first rectangle
PRECTL  prclRect2;    - Address of structure with second rectangle
```

WinInvalidateRect (Adds a rectangle to a window's update region)

```
#define INCL_WINWINDOWMGR

BOOL WinInvalidateRect(hwnd, prclPrc, fIncludeClippedChildren);

HWND    hwnd;                       - Handle of window with changed update region
PRECTL  prclPrc;                    - Address of structure with rectangle
BOOL    fIncludeClippedChildren;    - Invalidation-scope flag
```

WinInvalidateRegion (Adds a region to a window's update region)

```
#define INCL_WINWINDOWMGR

BOOL WinInvalidateRegion(hwnd, hrgn, fIncludeClippedChildren);

HWND  hwnd;                       - Handle of window with changed update region
HRGN  hrgn;                       - Handle of the region to add
BOOL  fIncludeClippedChildren;    - Invalidation-scope flag
```

WinInvertRect (Inverts a rectangular area)

```
#define INCL_WINRECTANGLES

BOOL WinInvertRect(hps, prclRect);

HPS     hps;          - Handle of the presentation space
PRECTL  prclRect;     - Address of structure with rectangle to invert
```

WinIsChild (Tests whether one window is descendant of another)

```
#define INCL_WINWINDOWMGR

BOOL WinIsChild(hwnd, hwndParent);

HWND  hwnd;           - Handle of the child window
HWND  hwndParent;     - Handle of the parent window
```

WinIsControlEnabled (Returns state of the item in the dialog)

```
#define INCL_WINWINDOWMGR

BOOL WinIsControlEnabled(hwndDlg, usId);

HWND   hwndDlg;     - Dialog window handle
SHORT  usId;        - Identity of the specified item
```

WinIsMenuItemChecked (Returns the state of the identified menu)

```
#define INCL_WINWINDOWMGR

BOOL WinIsMenuITemChecked(hwndMenu, usId);

HWND    hwndMenu;     - Menu window handle
USHORT  usId;         - Identity of the menu item
```

WinIsMenuItemEnabled (Returns the state of the identified menu)

```
#define INCL_WINWINDOWMGR

BOOL WinIsMenuITemEnabled(hwndMenu, usId);

HWND    hwndMenu;     - Menu window handle
USHORT  usId;         - Identity of the menu item
```

WinIsMenuItemValid (Returns TRUE if the item is valid choice)

```
#define INCL_WINWINDOWMGR

BOOL WinIsMenuITemValid(hwndMenu, usId);

HWND    hwndMenu;     - Menu window handle
USHORT  usId;         - Identity of the menu item
```

WinIsPhysInputEnabled (Determines whether physical input is enabled)

```
#define INCL_WININPUT

BOOL WinIsPhysInputEnabled(hwndDeskTop);

HWND  hwndDeskTop;     - Handle of the Desktop
```

WinIsRectEmpty (Determines whether a rectangle is empty)

```
#define INCL_WINRECTANGLES

BOOL WinIsRectEmpty(hab, prclprc);

HAB     hab;         - Handle of the anchor block
PRECTL  prclprc;     - Pointer to rectangle to check
```

WinIsThreadActive (Determines thread ownership)

```
#define INCL_WINWINDOWMGR

BOOL WinIsThreadActive(hab);

HAB  hab;     - Handle of the anchor block
```

WinIsWindow (Determines whether a window is valid)

```
#define INCL_WINWINDOWMGR

BOOL WinIsWindow(hab, hwnd);

HAB   hab;      - Handle of the anchor block
HWND  hwnd;     - Handle of window to test
```

WinIsWindowEnabled (Determines whether a window is enabled)

```
#define INCL_WINWINDOWMGR

BOOL WinIsWindowEnabled(hwnd);

HWND  hwnd;     - Handle of window to check
```

WinIsWindowShowing (Determines whether any part of window is showing)

```
#define INCL_WINWINDOWMGR

BOOL WinIsWindowShowing(hwnd);

HWND hwnd;     - Window handle
```

WinIsWindowVisible (Determines whether a window is visible)

```
#define INCL_WINWINDOWMGR

BOOL WinIsWindowVisible(hwnd);

HWND  hwnd;     - Handle of the window to test
```

WinLoadAccelTable (Loads an accelerator table)

```
#define INCL_WINACCELERATORS

HACCEL WinLoadAccelTable(hab, Resource, idAccelTable);

HAB      hab;             - Handle of the anchor block
HMODULE  Resource;        - Resource identity
ULONG    idAccelTable;    - Accelerator table identifier
```

WinLoadDlg (Creates a dialog window from a resource)

```
#define INCL_WINDIALOGS

HWND WinLoadDlg(hwndParent, hwndOwner, pfnDlgProc, hmod, idDlg, pCreateParams);

HWND     hwndParent;       - Handle of the parent window
HWND     hwndOwner;        - Handle of the owner window
PFNWP    pfnDlgProc;       - Pointer to the dialog procedure
HMODULE  hmod;             - Handle of resource with dialog template
ULONG    idDlg;            - Dialog window and template
PVOID    pCreateParams;    - Address of dialog-procedure data
```

WinLoadFileIcon (Returnes a pointer to a icon)

```
#define INCL_WINWORKPLACE

HPOINTER WinLoadFileIcon(pszFileName, fPrivate);

PSZ   pszFileName;     - Pointer to a file name whose icon will be loaded
BOOL  fPrivate;        - Icon usage flag
```

WinLoadHelpTable (Loads a help table)

```
#define INCL_WINHELP

BOOL WinLoadHelpTable(hwndHelpInstance, idHelpTable, Module);

HWND     hwndHelpInstance;     - Handle of help instance
ULONG    idHelpTable;          - Resource ID for help table
HMODULE  Module;               - Resource-module handle
```

WinLoadLibrary (Loads a dynamic-link library)

```
#define INCL_WINLOAD

HLIB WinLoadLibrary(hab, pszLibName);

HAB  hab;           - Anchor-block handle
PSZ  pszLibName;    - Pointer to library name
```

WinLoadMenu (Creates a menu window from a resource)

```
#define INCL_WINMENUS

HWND WinLoadMenu(hwndOwner, Resource, ulMenuid);

HWND     hwndOwner;     - Handle of the frame window
HMODULE  Resource;      - Handle of the module with resource
ULONG    ulMenuid;      - Menu template identifier
```

WinLoadMessage (Loads a message from a resource)

```
#define INCL_WINWINDOWMGR

SHORT WinLoadMessage(hab, hmodMod, ulid, lcchMax, pszBuffer);

HAB      hab;           - Handle of the anchor block
HMODULE  hmodMod;       - Module handle
ULONG    ulid;          - Message identifier
LONG     lcchMax;       - Buffer size
PSZ      pszBuffer;     - Address of buffer for message
```

WinLoadPointer (Loads a pointer resource)

```
#define INCL_WINPOINTERS

HPOINTER WinLoadPointer(hwndDeskTop, Resource, idPointer);

HWND     hwndDeskTop;     - Handle of the Desktop
HMODULE  Resource;        - Handle of the module with the resource
ULONG    idPointer;       - Resource identifier
```

WinLoadProcedure (Loads a window procedure from a library)

```
#define INCL_WINLOAD

PFNWP WinLoadProcedure(hab, hlibLibhandle, pszProcName);

HAB   hab;              - Anchor-block handle
HLIB  hlibLibhandle;    - Handle of library
PSZ   pszProcName;      - Pointer to procedure name
```

WinLoadString (Loads a string from a resource)

```
#define INCL_WINWINDOWMGR

LONG WinLoadString(hab, Resource, idString, lBufferMax, pszBuffer);

HAB      hab;           - Handle of the anchor block
HMODULE  Resource;      - Handle of the module with the string
ULONG    idString       - String identifier
LONG     lbufferMax;    - Size of the buffer
PSZ      pszBuffer;     - Address of the buffer for the string
```

WinLockPointerUpdate (Changes mouse pointer into a symbol)

```
#define INCL_WINPOINTERS

BOOL WinLockPointerUpdate(hwndDeskTop, hptrNew, ulTimeInterval);

HWND      hwndDeskTop;      - Handle of the Desktop window
HPOINTER  hptrNew;          - Handle pointer
ULONG     ulTimeInterval    - Lock out time
```

WinLockupSystem (Locks up the system)

```
#define INCL_WINMESSAGEMGR

BOOL WinLockupSystem(hab);

HAB  hab;    - Handle of the anchor block
```

WinLockVisRegions (Locks/unlocks visible regions of all windows)

```
#define INCL_WINWINDOWMGR

BOOL WinLockVisRegions(hwndDeskTop, fLock);

HWND  hwndDeskTop;    - Handle of the Desktop
BOOL  fLock;          - Lock/unlock flag
```

WinLockWindowUpdate (Prevents/allows drawing to a window)

```
#define INCL_WINWINDOWMGR

BOOL WinLockWindowUpdate(hwndDeskTop, hwndLockUpdate);

HWND  hwndDeskTop;       - Handle of the Desktop
HWND  hwndLockUpdate;    - Handle of the window to lock/unlock
```

WinMakePoints (Converts points to grsphic points)

```
#define INCL_WINRETANGLES

BOOL WinMakePoints(hab, pwptppt, cCount);

HAB      hab;         - Anchor-block handle
PWPOINT  pwptppt;     - Points to be converted
ULONG    cCount;      - Number of points to be converted
```

WinMakeRect (Converts a rectangle to a graphic rectangle)

```
#define INCL_WINRECTANGLES

BOOL WinMakeRect(hab, pwrcprc);

HAB    hab;         - Anchor-block handle
WRECT  pwrcprc;     - Rectangle to be converted
```

WinMapDlgPoints (Converts coordinates between window and a dialog)

```
#define INCL_WINDIALOGS

BOOL WinMapDlgPoints(hwndDlg, aptlPoints, ulCount, flOptions);

HWND     hwndDlg;        - Handle of the dialog window
PPOINTL  aptlPoints;     - Array of structures with points to map
ULONG    ulCount;        - Number of POINTL structures
BOOL     flOptions;      - Type of points
```

WinMapWindowPoints (Converts set of points between coordinate spaces)

```
#define INCL_WINWINDOWMGR

BOOL WinMapWindowPoints(hwndFrom, hwndTo, aptlPoints, lCount);

HWND     hwndFrom;       - Handle of the window to be mapped from
HWND     hwndTo;         - Handle of the window to be mapped to
PPOINTL  aptlPoints;     - Address of array of structures with points to map
LONG     lCount;         - Number of POINTL structures
```

WinMessageBox (Creates, displays, and operates a message box)

```
#define INCL_WINDIALOGS

USHORT WinMessageBox(hwndParent, hwndOwner, pszText, pszTitle, usWindow, flStyle);

HWND    hwndParent;     - Handle of the parent window
HWND    hwndOwner;      - Handle of the owner window
PSZ     pszText;        - Address of text in message box
PSZ     pszTitle;       - Address of title of message box
USHORT  usWindow;       - Message-box identifier
ULONG   flStyle;        - Type of message box
```

WinMultWindowFromIDs (Finds the handles of child windows)

```
#define INCL_WINWINDOWMGR

LONG WinMultWindowFromIDs(hwndParent, ahwnd, ulFirst, ulLast);

HWND   hwndParent;     - Handle of the parent window
PHWND  ahwnd;          - Address of array of window handles
ULONG  ulFirst;        - First window identifier in range
ULONG  ulLast;         - Last window identifier in range
```

WinNextChar (Moves to the next character in a string)

```
#define INCL_WINCOUNTRY

PSZ WinNextChar(hab, ulCodePage, ulCountry, pszCurrentChar);

HAB    hab;              - Handle of the anchor block
ULONG  ulCodePage;       - Code page
ULONG  ulCountry;        - Country code
PSZ    pszCurrentChar;   - Address of character in string
```

WinOffsetRect (Offsets a rectangle)

```
#define INCL_WINRECTANGLES

BOOL WinOffsetRect(hab, prclRect, lcx, lcy);

HAB     hab;         - Handle of the anchor block
PRECTL  prclRect;    - Address of the structure with rectangle
LONG    lcx;         - Horizontal lOffsetset
LONG    lcy;         - Vertical lOffsetset
```

WinOpenClipbrd (Opens the Clipboard)

```
#define INCL_WINCLIPBOARD

BOOL WinOpenClipbrd(hab);

HAB  hab;    - Handle of the anchor block
```

WinOpenWindowDC (Opens a device context for a window)

```
#define INCL_WINWINDOWMGR

HDC WinOpenWindowDC(hwnd);

HWND  hwnd;    - Handle of the window
```

WinPeekMsg (Inspects the thread's message queue)

```
#define INCL_WINMESSAGEMGR

BOOL WinPeekMsg(hab, pqmsgmsg, hwndFilter, ulFirst, ulLast, flOptions);

HAB    hab;           - Handle of the anchor block
PQMSG  pqmsgmsg;      - Address of structure
HWND   hwndFilter;    - Handle of the filter window
ULONG  ulFirst;       - First message
ULONG  ulLast;        - Last message
ULONG  flOptions;     - Status of message in queue
```

WinPopupMenu (Displays a pop-up menu)

```
#define INCL_WINWINDOWMGR

BOOL WinPopupMenu(hwndParent, hwndOwner, hwndMenu, lx, ly, idItem, fsOptions);

HWND    hwndParent;     - Parent-window handle
HWND    hwndOwner;      - Owner-window handle
HWND    hwndMenu;       - Pop-up menu handle
LONG    lx;             - X-coordinate of pop-up menu position
LONG    ly;             - Y-coordinate of pop-up menu position
ULONG   idItem;         - Item identitiy
USHORT  fsOptions;      - Options
```

WinPostMsg (Posts message to specified window's message queue)

```
#define INCL_WINMESSAGEMGR

BOOL WinPostMsg(hwnd, ulMsgid, mpParam1, mpParam2);

HWND    hwnd;         - Handle of the window to post message to
ULONG   ulMsgid;      - Message
MPARAM  mpParam1;     - First message parameter
MPARAM  mpParam2;     - Second message parameter
```

WinPostQueueMsg (Posts message to any message queue in the system)

```
#define INCL_WINMESSAGEMGR

BOOL WinPostQueueMsg(hmq, ulMsgid, mpParam1, mpParam2);

HMQ     hmq;          - Handle of the message queue
ULONG   ulMsgid;      - Message
MPARAM  mpParam1;     - First message parameter
MPARAM  mpParam2;     - Second message parameter
```

WinPrevChar (Moves to the previous character in a string)

```
#define INCL_WINCOUNTRY

PSZ WinPrevChar(hab, ulCodePage, ulCountry, pszStart, pszCurrentChar);

HAB     hab;              - Handle of the anchor block
ULONG   ulCodePage;       - Code page
ULONG   ulCountry;        - Country code
PSZ     pszStart;         - Address of string with character
PSZ     pszCurrentChar;   - Address of character in string
```

WinProcessDlg (Processes modal dialog messages)

```
#define INCL_WINDIALOGS

ULONG WinProcessDlg(hwndDlg);

HWND  hwndDlg;     - Handle of the dialog queue
```

WinPtInRect (Determines whether a point is within a rectangle)

```
#define INCL_WINRECTANGLES

BOOL WinPtInRect(hab, prclRect, pptlPoint);

HAB      hab;          - Handle of the anchor block
PRECTL   prclRect;     - Address of structure with rectangle coordinates
PPOINTL  pptlPoint;    - Address of structure with point coordinates
```

WinQueryAccelTable (Determines an accelerator table)

```
#define INCL_WINACCELERATORS

HACCEL WinQueryAccelTable(hab, hwndFrame);

HAB   hab;          - Handle of the anchor block
HWND  hwndFrame;    - Handle of the frame window
```

WinQueryActiveWindow (Retrieves the handle to the active window)

```
#define INCL_WINWINDOWMGR

HWND WinQueryActiveWindow(hwndParent);

HWND  hwndParent;    - Parent window handle
```

WinQueryAnchorBlock (Retrieves an anchor-block handle)

```
#define INCL_WINWINDOWMGR

HAB WinQueryAnchorBlock(hwnd);

HWND  hwnd;    - Window handle
```

WinQueryAtomLength (Retrieves the length of an atom)

```
#define INCL_WINATOM

ULONG WinQueryAtomLength(hatomtblAtomTbl, atom);

HATOMTBL  hatomtblAtomTbl;    - Handle of the atom table
ATOM      atom;               - Atom
```

WinQueryAtomName (Retrieves the name of an atom)

```
#define INCL_WINATOM

ULONG WinQueryAtomName(hatomtblAtomTbl, atom, pszBuffer, ulBufferMax);

HATOMTBL  hatomtblAtomTbl;    - Handle of the atom table
ATOM      atom;               - Atom
PSZ       pszBuffer;          - Address of the buffer
ULONG     ulBufferMax;        - Length of the buffer
```

WinQueryAtomUsage (Retrieves the usage count of an atom)

```
#define INCL_WINATOM

ULONG WinQueryAtomUsage(hatomtblAtomTbl, atom);

HATOMTBL  hatomtblAtomTbl;     - Handle of the atom table
ATOM      atom;                - Atom
```

WinQueryButtonCheckstate (Returns the checked state of button control)

```
#define INCL_WINWINDOWMGR

USHORT WinQueryButtonCheckstate(hwndDlg, usId);

HWND    hwndDlg;     - Dialog window handle
USHORT  usId;        - Button control identity
```

WinQueryCapture (Retrieves handle to the window with mouse capture)

```
#define INCL_WININPUT

HWND WinQueryCapture(hwndDeskTop, fLock);

HWND  hwndDeskTop;     - Desktop handle
```

WinQueryClassInfo (Retrieves window-class information)

```
#define INCL_WINWINDOWMGR

BOOL WinQueryClassInfo(hab, pszClassName, pclsiClassInfo);

HAB         hab;               - Handle of the anchor block
PSZ         pszClassName;      - Address of the class name
PCLASSINFO  pclsiClassInfo;    - Address of structure for class information
```

WinQueryClassName (Retrieves the name of a window class)

```
#define INCL_WINWINDOWMGR

LONG WinQueryClassName(hwnd, lLength, pchBuffer);

HWND  hwnd;          - Handle of the window
LONG  lLength;       - Length of the buffer
PCH   pchBuffer;     - Address of the buffer
```

WinQueryClassThunkProc (Queries the pointer-conversion procedure)

```
#define INCL_WINTHUNKAPI

PFN WinQueryClassThunkProc(pszClassName);

PSZ  pszClassName;     - Window-class name
```

WinQueryClipbrdData (Retrieves a handle to the current clipboard)

```
#define INCL_WINCLIPBOARD

ULONG WinQueryClipbrdData(hab, ulfmt);

HAB    hab;       - Handle of the anchor block
ULONG  ulfmt;     - Specifies the format of the data
```

WinQueryClipbrdFmtInfo (Retrieves the format of data on the clipboard)

```
#define INCL_WINCLIPBOARD

BOOL WinQueryClipbrdFmtInfo(hab, ulfmt, pulFmtInfo);

HAB     hab;           - Handle of the anchor block
ULONG   ulfmt;         - Specifies data format
PULONG  pulFmtInfo;    - Receives memory model and usage flags
```

WinQueryClipbrdOwner (Retrieves a handle to the clipboard owner window)

```
#define INCL_WINCLIPBOARD

HWND WinQueryClipbrdOwner(hab);

HAB  hab;    - Anchor-block handle
```

WinQueryClipbrdViewer (Retrieves a handle to the clipboard viewer window)

```
#define INCL_WINCLIPBOARD

HWND WinQueryClipbrdViewer(hab);

HAB  hab;    - Anchor-block handle
```

WinQueryCp (Retrieves the queue code-page identifier)

```
#define INCL_WINCOUNTRY

ULONG WinQueryCp(hmq);

HMQ  hmq;    - Handle of the message queue
```

WinQueryCpList (Retrieves identifiers of available code pages)

```
#define INCL_WINCOUNTRY

ULONG WinQueryCpList(hab, ulCount, aulCodePage);

HAB     hab;            - Handle of the anchor block
ULONG   ulCount;        - Maximum number of code pages to retrieve
PULONG  aulCodePage;    - Address of array to receive code pages
```

WinQueryCursorInfo (Retrieves information about the current cursor)

```
#define INCL_WINCURSORS

BOOL WinQueryCursorInfo(hwndDeskTop, pcsriCursorInfo);

HWND         hwndDeskTop;        - Handle of the Desktop
PCURSORINFO  pcsriCursorInfo;    - Address of structure for cursor information
```

WinQueryDesktopBkgnd (Retrieves current state of Desktop background)

```
#define INCL_WINWINDOWMGR

BOOL WinQueryDesktopBkgnd(hwndDeskTop, pDeskTopState);

HWND      hwndDeskTop;      - Handle of the Desktop
PDESKTOP  pDeskTopState;    - Address of DESKTOP structure
```

WinQueryDesktopWindow (Retrieves a handle to the Desktop window)

```
#define INCL_WINWINDOWMGR

HWND WinQueryDesktopWindow(hab, hdc);

HAB  hab;     - Handle of the anchor block
HDC  hdc;     - Handle of the device context
```

WinQueryDlgItemShort (Translates the text of a dialog item)

```
#define INCL_WINDIALOGS

BOOL WinQueryDlgItemShort(hwndDlg, idItem, psResult, fSigned);

HWND    hwndDlg;     - Handle of the dialog box
ULONG   idItem;      - Dialog-item identifier
PSHORT  psResult;    - Address of variable for result
BOOL    fSigned;     - Signed/unsigned flag
```

WinQueryDlgItemText (Retrieves the text of a dialog item)

```
#define INCL_WINDIALOGS

ULONG WinQueryDlgItemText(hwndDlg, idItem, lMaxText, pszText);

HWND   hwndDlg;      - Handle of the dialog box
ULONG  idItem;       - Identifies the dialog item
LONG   lMaxText;     - Size of the buffer
PSZ    pszText;      - Address of the buffer
```

WinQueryDlgItemTextLength (Retrieves the length of text in a dialog item)

```
#define INCL_WINDIALOGS

LONG WinQueryDlgItemTextLength(hwndDlg, idItem);

HWND   hwndDlg;     - Handle of the dialog box
ULONG  idItem;      - Dialog-item identifier
```

WinQueryFocus (Retrieves a handle to the focus window)

```
#define INCL_WININPUT

HWND WinQueryFocus(hwndDeskTop);

HWND  hwndDeskTop;     - Desktop handle
```

WinQueryHelpInstance (Finds a help instance)

```
#define INCL_WINHELP

HWND WinQueryHelpInstance(hwndApp);

HWND  hwndApp;     - Handle of application window
```

WinQueryLboxCount (Retrieves the number of items in the list box)

```
#define INCL_WINWINDOWMGR

SHORT WinQueryLboxCount(hwndLbox);

HWND  hwndLbox;     - List box handle
```

WinQueryLboxItemText (Fills the buffer with the text of indexed item)

```
#define INCL_WINWINDOWMGR

SHORT WinQueryLboxItemText(hwndLbox, sIndex, pszText, scchMax);

HWND   hwndLbox;     - List box handle
SHORT  sIndex;       - Index of the listbox item
PSZ    pszText;      - Pointer to text
SHORT  scchMax;      - Maximum number of the text
```

WinQueryLboxItemTextLength (Retrieves the length of the text of indexed item)

```
#define INCL_WINWINDOWMGR

SHORT WinQueryLboxItemTextLength(hwndLbox, sIndex);

HWND   hwndLbox;     - Listbox handle
SHORT  sIndex;       - Index of the item in the list box
```

WinQueryLboxSelectedItem (Returns the index of the selected item)

```
#define INCL_WINWINDOWMGR

SHORT WinQueryLboxSelectedItem(hwndLbox);

HWND  hwndLbox;     - List box handle
```

WinQueryMsgPos (Retrieves the position of a message)

```
#define INCL_WINMESSAGEMGR

BOOL WinQueryMsgPos(hab, pptlptrpos);

HAB      hab;           - Handle of the anchor block
PPOINTL  pptlptrpos;    - Address of structure for pointer position
```

WinQueryMsgTime (Retrieves the time at which a message was posted)

```
#define INCL_WINMESSAGEMGR

ULONG WinQueryMsgTime(hab);

HAB  hab;     - Handle of the anchor block
```

WinQueryObject (Retrieves a handle to the Desktop object window)

```
#define INCL_WINWORKPLACE

HWND WinQueryObject(pszObjectID);

PSZ pszObjectID;     - Pointer to object ID
```

WinQueryObjectWindow (Retrieves a handle to the Desktop object window)

```
#define INCL_WINWINDOWMGR

HWND WinQueryObjectWindow(hwndDeskTop);

HWND  hwndDeskTop;     - Handle of the Desktop
```

WinQueryPointer (Retrieves a handle to the mouse pointer)

```
#define INCL_WINPOINTERS

HPOINTER WinQueryPointer(hwndDeskTop);

HWND  hwndDeskTop;     - Handle of the Desktop
```

WinQueryPointerInfo (Retrieves mouse-pointer information)

```
#define INCL_WINPOINTERS

BOOL WinQueryPointerInfo(hptr, pptriPointerInfo);

HPOINTER      hptr;               - Handle of the pointer
PPOINTERINFO  pptriPointerInfo;   - Address of structure for pointer information
```

WinQueryPointerPos (Retrieves the position of the mouse pointer)

```
#define INCL_WINPOINTERS

BOOL WinQueryPointerPos(hwndDeskTop, pptlPoint);

HWND     hwndDeskTop;     - Handle of Desktop window
PPOINTL  pptlPoint;       - Address of structure for pointer position
```

WinQueryPresParam (Queries presentation parameters)

```
#define INCL_WINSYS

ULONG WinQueryPresParam(hwnd, idAttrType1, idAttrType2, pidAttrTypeFound,
                        cbAttrValueLen, pAttrValue, flOptions);

HWND    hwnd;                - Window handle
ULONG   idAttrType1;         - First parameter type to retrieve
ULONG   idAttrType2;         - Second parameter type to retrieve
PULONG  pidAttrTypeFound;    - Pointer to variable for parameter ID
ULONG   cbAttrValueLen;      - Buffer length
PVOID   pAttrValue;          - Pointer to buffer for presentation parameter
ULONG   flOptions;           - Flags
```

WinQueryQueueInfo (Retrieves queue information)

```
#define INCL_WINMESSAGEMGR

BOOL WinQueryQueueInfo(hmq, pmqiMsgInfo, cbCopied);

HMQ      hmq;            - Handle of the message queue
PMQINFO  pmqiMsgInfo;    - Address of structure for queue information
ULONG    cbCopied;       - Number of bytes of information to copy
```

WinQueryQueueStatus (Retrieves the status of a message queue)

```
#define INCL_WINMESSAGEMGR

ULONG WinQueryQueueStatus(hwndDeskTop);

HWND  hwndDeskTop;     - Handle of the Desktop
```

WinQuerySessionTitle (Retrieves the application's title)

```
#define INCL_WINSWITCHLIST

ULONG WinQuerySessionTitle(hab, ulSession, pszTitle, ulTitleLen)

HAB    hab;           - Anchor-block handle
ULONG  ulSession;     - Screen session
PSZ    pszTitle;      - Pointer to buffer for title
ULONG  ulTitleLen;    - Buffer length
```

WinQuerySwitchEntry (Retrieves a copy of the Window List data)

```
#define INCL_WINSWICHLIST

ULONG WinQuerySwitchEntry(hswitchSwitch, pswctlSwitchData);

HSWITCH   hswitchSwitch;       - Item handle
PSWCNTRL  pswctlSwitchData;    - Point to structure with item data
```

WinQuerySwitchHandle (Retrieves the switch-list handle)

```
#define INCL_WINSWICHLIST

HSWITCH WinQuerySwitchHandle(hwnd, idProcess);

HWND  hwnd;          - Window handle
PID   idProcess;     - Process identifier
```

WinQuerySwitchList (Obtains information from the Window List)

```
#define INCL_WINSWICHLIST

ULONG WinQuerySwitchList(hab, pswblkBlock, ulLength);

HAB       hab;            - Anchor-block handle
PSWBLOCK  pswblkBlock;    - Pointer to structure for items
ULONG     ulLength;       - Structure length
```

WinQuerySysColor (Retrieves a system color)

```
#define INCL_WINSYS

LONG WinQuerySysColor(hwndDeskTop, lColor, lReserved);

HWND  hwndDeskTop;     - Handle of the Desktop
LONG  lColor;          - Color index of color to retrieve
LONG  lReserved;       - Reserved
```

WinQuerySysModalWindow (Retrieves the system-modal window)

```
#define INCL_WINWINDOWMGR

HWND WinQuerySysModalWindow(hwndDeskTop);

HWND  hwndDeskTop;     - Handle of the Desktop
```

WinQuerySysPointer (Retrieves a handle to the system pointer)

```
#define INCL_WINPOINTERS

HPOINTER WinQuerySysPointer(hwndDeskTop, lIdentifier, fCopy);

HWND  hwndDeskTop;     - Handle of the Desktop
LONG  lIdentifier;     - System-pointer identifier
BOOL  fCopy;           - Load/unload flag
```

WinQuerySysPointerData (Retrieves icond data for the system pointer)

```
#define INCL_WINPOINTERS

HPOINTER WinQuerySysPointerData(hwndDeskTop, iptr, pIconInfo);

HWND  hwndDeskTop;     - Handle of the Desktop
LONG  iptr;            - Index of system pointer
BOOL  pIconInfo;       - Pointer to icon data buffer
```

WinQuerySystemAtomTable (Retrieves a handle to the system atom table)

```
#define INCL_WINATOM

HATOMTBL WinQuerySystemAtomTable(VOID);
```

WinQuerySysValue (Retrieves a system value)

```
#define INCL_WINSYS

LONG WinQuerySysValue(hwndDeskTop, iSysValue);

HWND  hwndDeskTop;     - Handle of Desktop
LONG  iSysValue;       - System value to retrieve
```

WinQueryTaskSizePos (Retrieves size, position, and status for window)

```
#define INCL_WINSWITCHLIST

ULONG WinQueryTaskSizePos(hab, ulID, pswp);

HAB    hab;      - Anchor-block handle
ULONG  ulID;     - Screen session
PSWP   pswp;     - Pointer to structure for defaults
```

WinQueryTaskTitle (Retrieves an application title)

```
#define INCL_WINSWITCHLIST

ULONG WinQueryTaskTitle(ulSession, pszTitle, ulTitleLen);

ULONG  ulSession;      - Session identity
PSZ    pszTitle;       - Address of the buffer
ULONG  ulTitleLen;     - Length of the buffer
```

WinQueryUpdateRect (Retrieves an update rectangle of a window)

```
#define INCL_WINWINDOWMGR

BOOL WinQueryUpdateRect(hwnd, prclPrc);

HWND    hwnd;        - Handle of the window
PRECTL  prclPrc;     - Address of structure for update rectangle
```

WinQueryUpdateRegion (Retrieves the update region of a window)

```
#define INCL_WINWINDOWMGR

LONG WinQueryUpdateRegion(hwnd, hrgn);

HWND  hwnd;     - Handle of the window
HRGN  hrgn;     - Handle of the region
```

WinQueryVersion (Retrieves the version and revision numbers)

```
#define INCL_WINWINDOWMGR

ULONG WinQueryVersion(hab);

HAB  hab;     - Handle of the anchor block
```

WinQueryWindow (Retrieves the handle to a window)

```
#define INCL_WINWINDOWMGR

HWND WinQueryWindow(hwnd, lCode);

HWND  hwnd;      - Handle of the window
LONG  lCode;     - Window to retrieve
```

WinQueryWindowDC (Retrieves the device context of a window)

```
#define INCL_WINWINDOWMGR

HDC WinQueryWindowDC(hwnd);

HWND  hwnd;     - Handle of the window
```

WinQueryWindowModel (Retrieves the memory model of the window)

```
#define INCL_WINTHUNKAPI

HDC WinQueryWindowModel(hwnd);

HWND  hwnd;     - Handle of the window
```

WinQueryWindowPos (Retrieves the size and position of a window)

```
#define INCL_WINWINDOWMGR

BOOL WinQueryWindowPos(hwnd, pswp);

HWND  hwnd;     - Handle of the window
PSWP  pswp;     - Address of the structure for window information
```

WinQueryWindowProcess (Retrieves process and thread identifiers)

```
#define INCL_WINWINDOWMGR

BOOL WinQueryWindowProcess(hwnd, pidPID, pidThreadID);

HWND  hwnd;           - Handle of the window
PPID  pidPID;         - Address of variable for process identifier
PTID  pidThreadID;    - Address of variable for thread identifier
```

WinQueryWindowPtr (Retrieves a pointer from reserved memory)

```
#define INCL_WINWINDOWMGR

PVOID WinQueryWindowPtr(hwnd, index);

HWND  hwnd;      - Handle of the window
LONG  index;     - Index to the pointer
```

WinQueryWindowRect (Retrieves the coordinates of a window)

```
#define INCL_WINWINDOWMGR

BOOL WinQueryWindowRect(hwnd, prclRect);

HWND    hwnd;        - Handle of the window
PRECTL  prclRect;    - Address of structure for window coordinates
```

WinQueryWindowText (Copies window text into a buffer)

```
#define INCL_WINWINDOWMGR

LONG WinQueryWindowText(hwnd, lLength, pchBuffer);

HWND  hwnd;          - Handle of the window
LONG  lLength;       - Length of the buffer
PCH   pchBuffer;     - Address of the buffer
```

WinQueryWindowTextLength (Retrieves the length of window text)

```
#define INCL_WINWINDOWMGR

LONG WinQueryWindowTextLength(hwnd);

HWND  hwnd;     - Handle of the window
```

WinQueryWindowThunkProc (Queries the pointer-conversion procedure)

```
#define INCL_WINTHUNKAPI

PFN WinQueryWindowThunkProc(hwnd);

HWND  hwnd;     - Handle of the window
```

WinQueryWindowULong (Retrieves a ULONG value from reserved memory)

```
#define INCL_WINWINDOWMGR

ULONG WinQueryWindowULong(hwnd, index);

HWND  hwnd;     - Handle of the window
LONG  index;    - Index of value to retrieve
```

WinQueryWindowUShort (Retrieves an integer from reserved memory)

```
#define INCL_WINWINDOWMGR

ULONG WinQueryWindowUShort(hwnd, index);

HWND  hwnd;     - Handle of the window
LONG  index;    - Index of value to retrieve
```

WinRealizePalette (Maps a color palette to device colors)

```
#define INCL_WIN

LONG WinRealizePalette(hwnd, hps, pcclr);

HWND    hwnd;     - Window handle
HPS     hps;      - Presentation-space handle
PULONG  pcclr;    - Number of physical palette entries changed
```

WinRegisterClass (Registers a window class)

```
#define INCL_WINWINDOWMGR

BOOL WinRegisterClass(hab, pszClassName, pWndProc, flClassStyle, usExtra);

HAB     hab;             - Handle of anchor block
PSZ     pszClassName;    - Pointer to class name
PFNWP   pWndProc;        - Address of window procedure
ULONG   flClassStyle;    - Window-style flags
USHORT  usExtra;         - Amount of reserved data
```

WinRegisterObjectClass (Registers a workplace object class)

```
#define INCL_WINWORKPLACE

BOOL WinRegisterObjectClass(pszClassName, pszModname);

PSZ  pszClassName;   - Pointer to the name of object class
PSZ  pszModname;     - Pointer to the name of DLL
```

WinRegisterUserDataType (Registers a data type and defines its structure)

```
#define INCL_WINMESSAGEMGR

BOOL WinRegisterUserDataType(hab, lDatatype, lCount, asTypes);

HAB     hab;          - Anchor-block handle
LONG    lDatatype;    - Data type code to be defined
LONG    lCount;       - Number of elements
PSHORT  asTypes;      - Data type codes of structure component
```

WinRegisterUserMsg (Registers a user message and its parameters)

```
#define INCL_WINMESSGEMGR

BOOL WinRegisterUserMsg(hab, ulMsgid, lType1, lDir1, lType2, lDir2, lTyper);

HAB   hab;         - Anchor-block handle
LONG  ulMsgid;     - Message identifier
LONG  lType1;      - Data type of message parameter 1
LONG  lDir1;       - Direction of message parameter 1
LONG  lType2;      - Data type of message parameter 2
LONG  lDir2;       - Direction of message parameter 2
LONG  lTyper;      - Data type od message reply
```

WinReleaseHook (Releases an application hook from a hook chain)

```
#define INCL_WINHOOKS

BOOL WinReleaseHook(hab, hmq, lHook, pAddress, Module);

HAB      hab;          - Handle of the anchor block
HMQ      hmq;          - Handle of the message queue
LONG     lHook;        - Hook identifier
PFN      pAddress;     - Address of the hook procedure
HMODULE  Module;       - Handle of the module with hook procedure
```

WinReleasePS (Releases a cached presentation space)

```
#define INCL_WINBOOL

WinReleasePS(hps);

HPS  hps;    - Presentation-space handle
```

WinRemovePresParam (Removes a presentation parameter)

```
#define INCL_WINSYS

BOOL WinRemovePresParam(hwnd, idAttrType);

HWND   hwnd;          - Window handle
ULONG  idAttrType;    - Presentation parameter to remove
```

WinRemoveSwitchEntry (Removes an entry from the switch list)

```
#define INCL_WINSWITCHLIST

USHORT WinRemoveSwitchEntry(hswitchSwitch);

HSWITCH  hswitchSwitch;    - Handle of the switch list
```

WinReplaceObjctClass (Replaces a registered class with another class)

```
#define INCL_WINWORKPLACE

BOOL WinRelaplaceObjectClass(pszOldClassName, pszNewClassName, fReplace);

PSZ   pszOldClassName;    - Pointer to the name of object class being replaced
PSZ   pszNewClassName;    - Pointer to the name of object class replacing
BOOL  fReplace;           - Replace/undo replace
```

WinRequestMutexSem (Requests ownership of a mutex semaphore)

```
#define INCL_WINMESSAGEMGR

ULONG WinRequestMutexSem(hmtx, ulTimeout);

HMTX   hmtx;          - Handle of mutex semaphore to request
ULONG  ulTimeout;     - Timeout in milliseconds
```

WinRestoreWindowPos (Restores the size and position of the window)

```
#define INCL_WINWORKPLACE

BOOL WinRestoreWindowPos(pszAppName, pszKeyName, hwnd);

PSZ   pszAppName;     - Pointer to the application name
PSZ   pszKeyName;     - Pointer to the key name
HWND  hwnd;           - Window handle
```

WinSaveWindowPos (Associates an array of SWP structures)

```
#define INCL_WINFRAMEMGR

BOOL WinSaveWindowPos(hsvwp, aswpaswp, ccswp);

HSVWP  hsvwp;        - Identifier of the frame window repositioning process
PSWP   aswpaswp;     - Array of SWP structures
ULONG  ccswp;        - Count of SWP structures
```

WinScrollWindow (Scrolls the contents of a window rectangle)

```
#define INCL_WINWINDOWMGR

LONG WinScrollWindow(hwnd, ldx, ldy, prclScroll, prclClip, hrgnUpdateRgn,
                     prclUpdate,flOptions);

HWND    hwnd;              - Handle of the window to scroll
LONG    ldx;               - Amount of horizontal scrolling
LONG    ldy;               - Amount of vertical scrolling
PRECTL  prclScroll;        - Address of structure with scroll rectangle
PRECTL  prclClip;          - Address of structure with clip rectangle
HRGN    hrgnUpdateRgn;     - Handle of the update region
PRECTL  prclUpdate;        - Address of the structure for the update rectangle
ULONG   flOptions;         - Scrolling flags
```

WinSendDlgItemMsg (Sends a message to a dialog item)

```
#define INCL_WINDIALOGS

MRESULT WinSendDlgItemMsg(hwndDlg, IdItem, ulMsgid, mpParam1, mpParam2);

HWND    hwndDlg;      - Handle of the dialog box
ULONG   IdItem;       - Dialog-item identifier
ULONG   ulMsgid;      - Message
MPARAM  mpParam1;     - First message parameter
MPARAM  mpParam2;     - Second message parameter
```

WinSendMsg (Sends a message to a window)

```
#define INCL_WINMESSAGEMGR

MRESULT WinSendMsg(hwnd, ulMsgid, mpParam1, mpParam2);

HWND    hwnd;         - Handle of the receiving window
ULONG   ulMsgid;      - Message
MPARAM  mpParam1;     - First message parameter
MPARAM  mpParam2;     - Second message parameter
```

WinSetAccelTable (Sets an accelerator table)

```
#define INCL_WINACCELERATORS

BOOL WinSetAccelTable(hab, haccelAccel, hwndFrame);

HAB     hab;             - Handle of the anchor block
HACCEL  haccelAccel;     - Handle of the accelerator table
HWND    hwndFrame;       - Handle of the frame window
```

WinSetActiveWindow (Sets the active window)

```
#define INCL_WINWINDOWMGR

BOOL WinSetActiveWindow(hwndDeskTop, hwnd);

HWND  hwndDeskTop;     - Handle of the Desktop
HWND  hwnd;            - Handle of the window to make active
```

WinSetCapture (Captures all mouse messages)

```
#define INCL_WININPUT

BOOL WinSetCapture(hwndDeskTop, hwnd);

HWND  hwndDeskTop;     - Handle of the Desktop
HWND  hwnd;            - Handle of the window to receive all mouse messages
```

WinSetClassMsgInterest (Sets the message interest of a window class)

```
#define INCL_WINMESSAGEMGR

BOOL WinSetClassMSgInterest(hab, pszClassName, ulMsgClass, lControl);

HAB    hab;              - Anchor-block handle
PSZ    pszClassName;     - Window-class name
ULONG  ulMsgClass;       - Message class to have interest level set
LONG   lControl          - Interest identifier
```

WinSetClassThunkProc (Associates a pointer-conversion procedure)

```
#define INCL_WINTHUNKAPI

BOOL WinSetClassThunkProc(pszClassName, pthunkpr);

PSZ  pszClassName;     - Window-class name
PFN  pthunkpr;         - Pointer-conversion procedure identifier
```

WinSetClipbrdData (Puts data on the clipboard)

```
#define INCL_WINCLIPBOARD

BOOL WinSetClipbrdData(hab, ulh, ulfmt, flFmtInfo);

HAB    hab;          - Handle of the anchor block
ULONG  ulh;          - Handle
ULONG  ulfmt;        - Specifies the format
ULONG  flFmtInfo;    - Specifies the data type
```

WinSetClipbrdOwner (Sets the current clipboard owner window)

```
#define INCL_WINCLIPBOARD

BOOL WinSetClipbrdOwner(hab, hwnd);

HAB   hab;     - Handle of the anchor block
HWND  hwnd;    - Handle of the clipboard owner
```

WinSetClipbrdViewer (Sets the current clipboard viewer window)

```
#define INCL_WINCLIPBOARD

BOOL WinSetClipbrdViewer(hab, hwndNewClipViewer);

HAB   hab;                  - Handle of the anchor block
HWND  hwndNewClipViewer;    - Handle of the clipboard viewer
```

WinSetCp (Sets the queue code-page identifier)

```
#define INCL_WINCOUNTRY

BOOL WinSetCp(hmq, ulCodePage);

HMQ    hmq;          - Handle of the message queue
ULONG  ulCodePage;   - Code page
```

WinSetDesktopBkgnd (Sets the Desktop background)

```
#define INCL_WINWINDOWMGR

HBITMAP WinSetDesktopBkgnd(hwndDeskTop, pDeskTopState);

HWND      hwndDeskTop;      - Handle of the Desktop window
PDESKTOP  pDeskTopState;    - State of Desktop structure
```

WinSetDlgItemShort (Sets dialog text to an integer)

```
#define INCL_WINDIALOGS

BOOL WinSetDlgItemShort(hwndDlg, idItem, usValue, fSigned);

HWND    hwndDlg;    - Handle of the dialog box
ULONG   idItem;     - Dialog-item identifier
USHORT  usValue;    - Value to set
BOOL    fSigned;    - Signed/unsigned flag
```

WinSetDlgItemText (Sets the text in a dialog item)

```
#define INCL_WINDIALOG

BOOL WinSetDlgItemText(hwndDlg, idItem, pszText);

HWND   hwndDlg;     - Handle of the dialog box
ULONG  idItem;      - Dialog-item identifier
PSZ    pszText;     - Text to set
```

WinSetFileIcon (Sets the icon on the file to be that specified by picon)

```
#define INCL_WINWORKPLACE

BOOL WinSetFileIcon(pszFileName, picon);

PSZ        pszFileName;     - Pointer to the name of the file
PICONINFO  picon;           - Pointer to ICONINFO structure
```

WinSetFocus (Sets the focus window)

```
#define INCL_WININPUT

BOOL WinSetFocus(hwndDeskTop, hwndNewFocus);

HWND  hwndDeskTop;      - Handle of the Desktop
HWND  hwndNewFocus;     - Handle of the window receiving the focus
```

WinSetHook (Installs a hook)

```
#define INCL_WINHOOKS

BOOL WinSetHook(hab, hmq, lHookType, pHookProc, Module);

HAB      hab;           - Handle of the anchor block
HMQ      hmq;           - Handle of the message queue
LONG     lHookType;     - Type of hook chain
PFN      pHookProc;     - Address of the hook procedure
HMODULE  Module;        - Handle of the module with the hook procedure
```

WinSetKeyboardStateTable (Gets or sets the keyboard state)

```
#define INCL_WININPUT

BOOL WinSetKeyboardStateTable(hwndDeskTop, abKeyStateTable, fSet);

HWND   hwndDeskTop;         - Handle of the Desktop
PBYTE  abKeyStateTable;     - Address of the key table
BOOL   fSet;                - Set/copy flag
```

WinSetLboxItemText (Sets the text of an indexed list box item to the buffer)

```
#define INCL_WINWINDOWMGR

BOOL WinSetLboxItemText(hwndLbox, sLboxIndx, pszText);

HWND   hwndLbox;      - Handle of the list box
SHORT  sLboxIndx;     - Index of the list box item
PSZ    pszText;       - Pointer to a null-terminated string
```

WinSetMenuItemText (Sets the text of an indexed Menu item to the buffer)

```
#define INCL_WINWINDOWMGR

BOOL WinSetMenuItemText(hwndMenu, usid, pszText);

HWND   hwndMenu;     - Handle of the Desktop
SHORT  usid;         - Address of the key table
PSZ    pszText;      - Pointer to a null-terminated string
```

WinSetMsgInterest (Sets the message interest of a window)

```
#define INCL_WINMESSAGEMGR

BOOL WinSetMsgInterest(hwnd, ulMsgClass, lControl);

HWND   hwnd;          - Handle of the window
ULONG  ulMsgClass;    - Message class for which to set the interest level
LONG   lControl;      - Interest identifier
```

WinSetMsgMode (Sets the message mode for application's private window class)

```
#define INCL_WINMESSAGEMGR

BOOL WinSetMsgMode(hab, pszClassName, lControl);

HAB   hab;             - Handle of the anchor block
PSZ   pszClassName;    - Window class name
LONG  lControl;        - Message mode identifier
```

WinSetMultWindowPos (Sets multiple window positions)

```
#define INCL_WINWINDOWMGR

BOOL WinSetMultWindowPos(hab, pSwp, cCount);

HAB    hab;        - Handle of the anchor block
PSWP   pSwp;       - Address of array of SWP structures
ULONG  cCount;     - Number of SWP structures
```

WinSetObjectData (Sets data on a Workplace Shell object)

```
#define INCL_WINWORKPLACE

BOOL WinSetObjectData(object, pszSetupString);

HOBJECT  object;            - Handle of a Workplace Shell object
PSZ      pszSetupString;    - Pointer to a zero-terminated string
```

WinSetOwner (Changes the owner window)

```
#define INCL_WINWINDOWMGR

BOOL WinSetOwner(hwnd, hwndNewOwner);

HWND  hwnd;            - Handle of the window whose owner is changed
HWND  hwndNewOwner;    - Handle of the new owner window
```

WinSetParent (Changes the handle to the parent window)

```
#define INCL_WINWINDOWMGR

BOOL WinSetParent(hwnd, hwndNewParent, fRedraw);

HWND  hwnd;             - Handle of the window whose parent is changed
HWND  hwndNewParent;    - Handle of the new parent window
BOOL  fRedraw;          - Redraw flag
```

WinSetPointer (Sets the mouse pointer)

```
#define INCL_WINPOINTERS

BOOL WinSetPointer(hwndDeskTop, hptrNewPointer);

HWND      hwndDeskTop;       - Handle of the Desktop
HPOINTER  hptrNewPointer;    - Handle of the pointer
```

WinSetPointerPos (Sets the position of the mouse pointer)

```
#define INCL_WINPOINTERS

BOOL WinSetPointerPos(hwndDeskTop, lx, ly);

HWND  hwndDeskTop;     - Handle of the Desktop
LONG  lx;              - Horizontal position
LONG  ly;              - Vertical position
```

WinSetPresParam (Sets presentation parameters)

```
#define INCL_WINSYS

BOOL WinSetPresParam(hwnd, idAttrType, cbAttrValueLen, pAttrValue);

HWND   hwnd;               - Window handle
ULONG  idAttrType;         - Presentation parameter
ULONG  cbAttrValueLen;     - Presentation-parameter size
PVOID  pAttrValue;         - Pointer to presentation parameter
```

WinSetRect (Sets the coordinates of a rectangle)

```
#define INCL_WINRECTANGLES

BOOL WinSetRect(hab, prclRect, lLeft, lBottom, lRight, lTop);

HAB     hab;          - Handle of the anchor block
PRECTL  prclRect;     - Address of structure with rectangle to set
LONG    lLeft;        - Left side
LONG    lBottom;      - Bottom side
LONG    lRight;       - Right side
LONG    lTop;         - Top side
```

WinSetRectEmpty (Sets a rectangle to empty)

```
#define INCL_WINRECTANGLES

BOOL WinSetRectEmpty(hab, prclRect);

HAB     hab;          - Handle of the anchor block
PRECTL  prclRect;     - Address of structure with rectangle to set to empty
```

WinSetSyncroMode (Synchronizes processing of messages in a distributed queue)

```
#define INCL_WINMESSAGEMGR

BOOL WinSetSyncroMode(hab, lMode);

HAB   hab;       - Handle of the anchor block
LONG  lMode;     - Synchronization mode
```

WinSetSysColors (Sets the system colors)

```
#define INCL_WINSYS

BOOL WinSetSysColors(hwndDeskTop, flOptions, ulFormat, lStart, ulTablen, alTable);

HWND   hwndDeskTop;     - Handle of the Desktop
ULONG  flOptions;       - Color options
ULONG  ulFormat;        - Format options
LONG   lStart;          - Start system color index
ULONG  ulTablen;        - Number of colors to set
PLONG  alTable;         - Address of color definitions
```

WinSetSysModalWindow (Makes a window system modal)

```
#define INCL_WINWINDOWMGR

BOOL WinSetSysModalWindow(hwndDeskTop, hwnd);

HWND hwndDeskTop;     - Handle of the Desktop
HWND hwnd;            - Handle of the window that becomes system modal
```

WinSetSysValue (Sets a system value)

```
#define INCL_WINSYS

BOOL WinSetSysValue(hwndDeskTop, lValueid, lValue);

HWND  hwndDeskTop;     - Handle of Desktop window
LONG  lValueid;        - System value to change
LONG  lValue;          - New system value
```

WinSetWindowBits (Writes into a bit field)

```
#define INCL_WINWINDOWMGR

BOOL WinSetWindowBits(hwnd, lb, flData, flMask);

HWND   hwnd;       - Handle of the window
LONG   lb;         - Index of the bits
ULONG  flData;     - Data to set
ULONG  flMask;     - Mask of bits to set
```

WinSetWindowPos (Sets the size and position of a window)

```
#define INCL_WINWINDOWMGR

BOOL WinSetWindowPos(hwnd, hwndBehind, lx, ly, lcx, lcy, flOptions);

HWND   hwnd;          - Handle of window being set
HWND   hwndBehind;    - Placement-order handle
LONG   lx;            - Horizontal position
LONG   ly;            - Vertical position
LONG   lcx;           - Width
LONG   lcy;           - Height
ULONG  flOptions;     - Window-positioning flags
```

WinSetWindowPtr (Places a pointer in reserved memory)

```
#define INCL_WINWINDOWMGR

BOOL WinSetWindowPtr(hwnd, lbx, pp);

HWND   hwnd;     - Handle of the window
LONG   lb;       - Index of the reserved memory
PVOID  pp;       - Pointer to place into reserved memory
```

WinSetWindowText (Sets the window text)

```
#define INCL_WINWINDOWMGR

BOOL WinSetWindowText(hwnd, pszString);

HWND  hwnd;          - Handle of the window
PSZ   pszString;     - Pointer to the text to set
```

WinSetWindowThunkProc (Associates pointer-conversion process with window)

```
#define INCL_WINTHUNKAPI

BOOL WinSetWindowThunkProc(hwnd, pthunkpr);

HWND  hwnd;         - Handle of the window
PFN   pthunkpr;     - Pointer-conversion process indentifier
```

WinSetWindowULong (Places a ULONG value in reserved memory)

```
#define INCL_WINWINDOWMGR

BOOL WinSetWindowULong(hwnd, lb, ulData);

HWND   hwnd;       - Handle of the window
LONG   lb;         - Index into reserved memory
ULONG  ulData;     - Value to place in reserved memory
```

WinSetWindowUShort (Places an integer in reserved memory)

```
#define INCL_WINWINDOWMGR

BOOL WinSetWindowUShort(hwnd, lb, usData);

HWND    hwnd;       - Handle of the window
LONG    lb;         - Index into reserved memory
USHORT  usData;     - Value to place in reserved memory
```

WinShowCursor (Shows or hides the cursor)

```
#define INCL_WINCURSORS

BOOL WinShowCursor(hwnd, fShow);

HWND  hwnd;      - Handle of the window
BOOL  fShow;     - Show/hide flag
```

WinShowPointer (Shows or hides the mouse pointer)

```
#define INCL_WINPOINTERS

BOOL WinShowPointer(hwndDeskTop, fShow);

HWND  hwndDeskTop;     - Handle of the Desktop
BOOL  fShow;           - Show/hide flag
```

WinShowTrackRect (Shows or hides a tracking rectangle)

```
#define INCL_WINTRACKRECT

BOOL WinShowTrackRect(hwnd, fShow);

HWND  hwnd;      - Handle of the window
BOOL  fShow;     - Show/hide flag
```

WinShowWindow (Sets the visibility state of a window)

```
#define INCL_WINWINDOWMGR

BOOL WinShowWindow(hwnd, fNewVisibility);

HWND  hwnd;               - Handle of the window
BOOL  fNewVisibility;     - Show/hide flag
```

WinShutdownSystem (Closes the system)

```
#define INCL_WINWORKPLACE

BOOL WinShutdownSystem(hab, hmq);

HAB  hab;     - Handle of the anchor block
HMQ  hmq;     - Handle of the message queue
```

WinStartApp (Starts an application)

```
#define INCL_WINWINDOWMGR

HAPP WinStartApp(hwndNotify, pDetails, pszParams, pReserved, ulOptions);

HWND          hwndNotify;     - Handle of the notification-window
PPROGDETAILS  pDetails;       - Program list structure
PSZ           pszParams;      - Input parameters
PVOID         pReserved;      - Must be NULL
ULONG         ulOptions;      - Option indicators
```

WinStartTimer (Starts a timer)

```
#define INCL_WINTIMER

ULONG WinStartTimer(hab, hwnd, idTimer, ulTimeout);

HAB     hab;          - Handle of the anchor block
HWND    hwnd;         - Handle of the window
ULONG   idTimer;      - Timer identifier
ULONG   ulTimeout;    - Timeout value
```

WinStopTimer (Stops a timer)

```
#define INCL_WINTIMER

BOOL WinStopTimer(hab, hwnd, ulTimer);

HAB     hab;        - Handle of the anchor block
HWND    hwnd;       - Handle of the window
ULONG   ulTimer;    - Timer identifier
```

WinStoreWindowPos (Saves current size and position of window)

```
#define INCL_WINWORKPLACE

BOOL WinStoreWindowPos(pszAppName, pszKeyName, hwnd);

PSZ   pszAppName;     - Pointer to zero-terminated string
PSZ   pszKeyName;     - Pointer to zero-terminated string
HWND  hwnd;           - Handle to the window
```

WinSubclassWindow (Creates a subclass of a window)

```
#define INCL_WINWINDOWMGR

PFNWP WinSubclassWindow(hwnd, pNewWindowProc);

HWND   hwnd;              - Handle of the window to subclas
PFNWP  pNewWindowProc;    - Address of new window procedure
```

WinSubstituteStrings (Performs a string substitution)

```
#define INCL_WINDIALOGS

LONG WinSubstituteStrings(hwnd, pszSrc, lDestMax, pszDest);

HWND  hwnd;         - Handle of the window
PSZ   pszSrc;       - Address of the source string
LONG  lDestMax;     - Size of destination string buffer
PSZ   pszDest;      - Address of buffer for destination string
```

WinSubtractRect (Subtracts rectangles)

```
#define INCL_WINRECTANGLES

BOOL WinSubtractRect(hab, prclDest, prclSrc1, prclSrc2);

HAB     hab;          - Handle of the anchor block
PRECTL  prclDest;     - Address of the destination rectangle structure
PRECTL  prclSrc1;     - Address of the first rectangle structure
PRECTL  prclSrc2;     - Address of the second rectangle structure
```

WinSwitchToProgram (Makes an application the active application)

```
#define INCL_WINSWITCHLIST

ULONG WinSwitchToProgram(hswitchSwHandle);

HSWITCH  hswitchSwHandle;     - Handle of application to activate
```

WinTerminate (Terminates an application thread)

```
#define INCL_WINWINDOWMGR

BOOL WinTerminate(hab);

HAB  hab;     - Handle of the anchor block
```

WinTerminateApp (Terminates an application)

```
#define INCL_WINWINDOWMGR

BOOL WinTerminateApp(happ);

HAPP  happ;     - Handle to anchor block
```

WinTrackRect (Draws a tracking rectangle)

```
#define INCL_WINTRACKRECT

BOOL WinTrackRect(hwnd, hps, ptiTrackInfo);

HWND        hwnd;            - Handle of the window
HPS         hps;             - Handle of the presentation space
PTRACKINFO  ptiTrackInfo;    - Address of structure for tracking information
```

WinTranslateAccel (Translates a WM_CHAR message)

```
#define INCL_WINACCELERATORS

BOOL WinTranslateAccel(hab, hwnd, haccelAccel, pQmsg);

HAB     hab;            - Handle of the anchor block
HWND    hwnd;           - Handle of the window
HACCEL  haccelAccel;    - Handle of the accelerator table
PQMSG   pQmsg;          - Address of structure with message
```

WinUnionRect (Calculates a union rectangle)

```
#define INCL_WINRECTANGLES

BOOL WinUnionRect(hab, prclDest, prclSrc1, prclSrc2);

HAB     hab;          - Handle of the anchor block
PRECTL  prclDest;     - Address of the destination rectangle structure
PRECTL  prclSrc1;     - Address of the first rectangle structure
PRECTL  prclSrc2;     - Address of the second rectangle structure
```

WinUpdateWindow (Updates a window)

```
#define INCL_WINWINDOWMGR

BOOL WinUpdateWindow(hwnd);

HWND  hwnd;     - Handle of the window
```

WinUpper (Converts a string to uppercase)

```
#define INCL_WINCOUNTRY

ULONG WinUpper(hab, ulCodePage, ulCountry, pszString);

HAB    hab;           - Handle of the anchor block
ULONG  ulCodePage;    - Code-page identifier
ULONG  ulCountry;     - Country-code identifier
PSZ    pszString;     - Address of the string to convert
```

WinUpperChar (Converts a character to uppercase)

```
#define INCL_WINCOUNTRY

ULONG WinUpperChar(hab, ulCodePage, ulCountry, ulInchar);

HAB    hab;           - Handle of the anchor block
ULONG  ulCodePage;    - Code-page identifier
ULONG  ulCountry;     - Country-code identifier
ULONG  ulInchar;      - Character to translate
```

WinValidateRect (Validates a rectangle)

```
#define INCL_WINWINDOWMGR

BOOL WinValidateRect(hwnd, prclRect, fIncludeClippedChildren);

HWND    hwnd;                       - Handle of the window
PRECTL  prclRect;                   - Pointer to validation rectangle
BOOL    fIncludeClippedChildren;    - Inclusion flag
```

WinValidateRegion (Validates a region)

```
#define INCL_WINWINDOWMGR

BOOL WinValidateRegion(hwnd, hrgn, fIncludeClippedChildren);

HWND  hwnd;                       - Handle of the window
HRGN  hrgn;                       - Handle of the valid region
BOOL  fIncludeClippedChildren;    - Inclusion flag
```

WinWaitEventSem (Causes calling thread to wait until event semaphore posted)

```
#define INCL_WINMESSAGEMGR

ULONG WinWaitEventSem(hev, ulTimeout);

HEV    hev;          - Handle of the event semaphore
ULONG  ulTimeout;    - Timeout in milliseconds
```

WinWaitMsg (Waits for a filtered message)

```
#define INCL_WINMESSAGEMGR

BOOL WinWaitMsg(hab, ulFirst, ulLast);

HAB    hab;        - Handle of the anchor block
ULONG  ulFirst;    - First message
ULONG  ulLast;     - Last message
```

WinWaitMuxWaitSem (Waits for a muxwait semaphore to clear)

```
#define INCL_WINMESSAGEMGR

ULONG WinWaitMuxWaitSem(hmux, ulTimeout, pUser);

HMUX    hmux;          - Handle of the muxwait semaphore
ULONG   ulTimeout;     - Timeout in milliseconds
PULONG  pUser;         - Pointer to receive the user field of the semaphore
```

WinWindowFromDC (Retrieves a window handle from a device context)

```
#define INCL_WINWINDOWMGR

HWND WinWindowFromDC(hdc);

HDC  hdc;    - Handle of the device context
```

WinWindowFromID (Returns the handle to a child window)

```
#define INCL_WINWINDOWMGR

HWND WinWindowFromID(hwndParent, ulIdentifier);

HWND   hwndParent;      - Parent-window handle
ULONG  ulIdentifier;    - Window identifier
```

WinWindowFromPoint (Finds a window below a point)

```
#define INCL_WINWINDOWMGR

HWND WinWindowFromPoint(hwndParent, pptlPoint, fEnumChildren);

HWND     hwndParent;       - Handle of the window
PPOINTL  pptlPoint;        - Address of structure with the point
BOOL     fEnumChildren;    - Scope flag
```

Appendixes

Appendix A. Reusable Submodules

The following submodules can be used with little or no modification:

OS2F0001.C	Right justify text	Page 362
OS2F0002.C	Display database error	Page 362
OS2F0003.C	Get selected list box entry	Page 363
OS2F0004.C	Create window	Page 363
OS2F0006.C	Create window notebook	Page 365
OS2F0007.C	Insert notebook page	Page 366
OS2F0008.C	Create list box	Page 367
OS2F0010.C	Create container list	Page 368
OS2F0011.C	Set entry field limit	Page 369
OS2F0012.C	Toggle buttons	Page 369
OS2F0013.C	Check control button state	Page 370
OS2F0014.C	Initialize help message	Page 370
OS2F0015.C	Insert list box item	Page 372
OS2F0026.C	Destroy suballocated memory	Page 372

Figure A-1. Submodule OS2F0001.C

```
/*========================== Right Justify Text ==============================
FUNCTION:    RtJustCharString
DESCRIPTION: This function right-justifies a text string.
PARAMETERS:  1. Handle to presentation space
             2. Pointer to text position coordinates structure
             3. Length of text string
             4. Char far pointer to text string
RETURNS:     LONG - Result of GpiCharStringAt function
============================================================================*/
#define INCL_GPI
#include <os2.h>

LONG RtJustCharString (HPS hps, POINTL *ptl, LONG lgth, PSZ TextStr)
{
   POINTL xpos;
   POINTL aptlTextBox[TXTBOX_COUNT];


/*============================================================================
   /* The TXTBOX_CONCAT element of the aptl structure contains the coordinates
   /* of the end of the text string. Subtracting this from the x-field of
   /* the POINTL structure passed to this function, gives the x-coordinate
   /* that will result in right-justified text.

/* ==========================================================================

   xpos.x = ptl->x;
   xpos.y = ptl->y;
   GpiQueryTextBox (hps, lgth, TextStr, TXTBOX_COUNT, aptlTextBox);
   xpos.x -= aptlTextBox[TXTBOX_CONCAT].x;

   return(GpiCharStringAt(hps, &xpos, lgth, TextStr));
}
```

Figure A-2. Submodule OS2F0002.C

```
/*========================== Display Database Error ==========================
FUNCTION:    DisplayDBerror
DESCRIPTION: This function sets up the parameters and calls the user-message
             display function in the event of a database access error.
PARAMETERS:  Long int. - DB return code.
RETURNS:     Void
============================================================================*/
#include <os2.h>
#include <stdlib.h>
#include "winvmsg.h"

VOID DisplayDBerror(LONG ErrCd)
{
   USHORT fCritMsg = MB_OK | MB_MOVEABLE | MB_CUACRITICAL;
   CHAR   ChStr[20];

   _ltoa(ErrCd, ChStr, 10);
   DisplayMessage(IDS_WHINV_MSG_01, HWND_DESKTOP, fCritMsg, ChStr);
}
```

Figure A-3. Submodule OS2F0003.C

```
/*======================== Get Selected List Box Entry ========================
FUNCTION:    GetSelectedLBEntry
DESCRIPTION: Determine if a list box entry has been selected.
PARAMETERS:  Handle to the list box.
RETURNS:     Line index if one is selected; otherwise, LIT_NONE.
============================================================================*/
#define INCL_PM
#include <os2.h>

USHORT GetSelectedLBEntry(HWND ListBoxHandle)
{
   static SHORT Index;
   Index = SHORT1FROMMR(WinSendMsg (ListBoxHandle, LM_QUERYSELECTION,
                                    NULL, NULL));
   return(Index);
}
```

Figure A-4. Submodule OS2F0004.C

```
/*============================== Create Window ===============================
FUNCTION:    CreateWindow.
DESCRIPTION: This function creates a window (main window, child window, or
             icon window, etc.) and sets its initial size and position. It
             returns the handle to the frame window.
PARAMETERS:  1. Handle to the parent of the window to be created
             2. Frame control flags
             3. Window Class name
             4. Window title
             5. Resource id value
             6. Initial horizontal location of origin
             7. Initial vertical
             8. Initial width
             9. Initial height
            10. Handle to the client area of the window
            11. Frame window style
            12. User defined size and location flags
RETURNS:    Frame window handle - Successful
            NULL                - Unsuccessful
============================================================================*/
#define INCL_PM
#include <os2.h>

HWND CreateWindow
(
   HWND   hWndParent,  /* Handle to the parent of the window to be created  */
   ULONG  ctldata,     /* Frame control flags for the window                */
   CHAR   *appname,    /* Class name of the window                          */
   CHAR   *title,      /* Title of the window                               */
   USHORT ResID,       /* Resource id value                                 */
   INT    x,           /* Initial horizontal and vertical location of origin*/
   INT    y,
   INT    cx,          /* Initial width and height of the window            */
   INT    cy,
   PHWND  hWndClient,  /* Handle to the client area of the window           */
   ULONG  lfStyle,     /* Frame window style                                */
```

Submodule OS2F0004.C (continued)

```
   USHORT uSizeStyle)  /* User defined size and location flags              */
   USHORT rc;          /* Return code                                       */
   USHORT SizeStyle;   /* Local window positioning options                  */
   HWND   hWndFrame;   /* Frame window handle                               */

   /* Create the frame window */
   hWndFrame = WinCreateStdWindow(hWndParent,       /* Parent of window     */
                                  lfStyle,          /* Frame window style   */
                                  &ctldata,         /* Frame flags          */
                                  appname,          /* Class name           */
                                  title,            /* Window title         */
                                  0L,               /* Client window style  */
                                  0L,               /* Module for resources */
                                  ResID,            /* Resource id          */
                                  (HWND *)hWndClient);

   /* If hWndFrame is NULL, an error occurred when opening window, notify    */
   /* user and exit this function.                                           */

   if(hWndFrame == 0L)
   {
      WinMessageBox(HWND_DESKTOP,
                    hWndParent,
                    "Unable to open window",
                    0L, 0L,
                    MB_OK|MB_ICONEXCLAMATION);
      return((HWND)NULL);
   }

   /* Set up size options                                                    */
   SizeStyle = SWP_ACTIVATE | SWP_ZORDER | uSizeStyle;

   /* If either the initial x or y position of the window is non-zero,       */
   /* set the window to its new position.                                    */
   if ((x  0) || (y  0))
      SizeStyle |= SWP_MOVE;

   /* If either the width or the height are non-zero, the size of the        */
   /* created window will be changed, so set SizeStyle accordingly.    ..*/
   if ((cx  0) || (cy  0))
      SizeStyle |= SWP_SIZE;

   /* Set the size and position of the window and activate it                */
   rc = WinSetWindowPos(hWndFrame, HWND_TOP, x, y, cx, cy, SizeStyle);

   /* If rc is not TRUE, WinSetWindowPos failed; notify user and exit        */
   if(!rc)
   {
      WinMessageBox(HWND_DESKTOP,
                    hWndParent,
                    "Unable to position window",
                    0L, 0L,
                    MB_OK|MB_ICONEXCLAMATION);
      return((HWND)NULL);
   }
   return(hWndFrame);
}
```

Figure A-5. Submodule OS2F0006.C

```
/*========================= Create Window Notebook ============================
FUNCTION:    CreateNotebook
DESCRIPTION: Creates a notebook control in a client window.
PARAMETERS:  Client window handle.
             Pointer to NOTEBOOK_DATA structure.
RETURNS:     TRUE if error, FALSE otherwise. I/p structure updated with
             notebook handle.
============================================================================*/
#include <os2.h>
#include <stdlib.h>
#include "notebook.h"

USHORT CreateWinNoteBook(HWND hWndClient, NOTEBOOK_DATA *Notebook)
{
   Notebook-hWndNotebook = WinCreateWindow(hWndClient, WC_NOTEBOOK,
                                           (PSZ)NULL,
                                           Notebook-ulNotebookStyle,
                                           Notebook-xLeft, Notebook-yBottom,
                                           Notebook-xRight, Notebook-yTop,
                                           hWndClient, HWND_TOP,
                                           Notebook-NotebookID,
                                           (PVOID)NULL, (PVOID)NULL);

   if (Notebook-hWndNotebook == (HWND)NULL)
   {
      return 1;
   }
   else
   {
      WinShowWindow(Notebook-hWndNotebook, TRUE);
      return 0;
   }
}
```

Figure A-6. Submodule OS2F0007.C

```
/*========================= Insert Notebook Page ============================
FUNCTION:     InsertNotebookPage
DESCRIPTION:  Inserts a page into a notebook.
PARAMETERS:   Pointer to NOTEBOOK_DATA structure.
              Pointer to PAGE_DATA structure.
RETURNS:      TRUE if error, FALSE otherwise.
              I/p structure updated with notebook handle.
============================================================================*/
#define  INCL_WIN
#include <os2.h>
#include "notebook.h"

VOID InsertNotebookPage(NOTEBOOK_DATA *Notebook, PAGE_DATA *Page)
{
   if(Page-usPgPosn == BKA_FIRST || Page-usPgPosn == BKA_LAST)
   {
      Page-ulPageId = (ULONG) WinSendMsg(Notebook-hWndNotebook,
                                         BKM_INSERTPAGE,
                                         (MPARAM)NULL,
                                         MPFROM2SHORT(Page-usPageAttrib,
                                                      Page-usPgPosn));
   }
   else
   {
      Page-ulPageId = (ULONG) WinSendMsg(Notebook-hWndNotebook,
                                         BKM_INSERTPAGE,
                                         MPFROMLONG(Page-ulPageId),
                                         MPFROM2SHORT(Page-usPageAttrib,
                                                      Page-usPgPosn));
   }

   /* Set Status line and major/minor tab text, if present */

   if(Page-StatusText != (CHAR *)NULL)
   {
      WinSendMsg(Notebook-hWndNotebook,
                 BKM_SETSTATUSLINETEXT,
                 MPFROMLONG(Page-ulPageId),
                 Page-StatusText);
   }

   if(Page-TabText != (CHAR *)NULL)
   (
      WinSendMsg(Notebook-hWndNotebook,
                 BKM_SETTABTEXT,
                 MPFROMLONG(Page-ulPageId),
                 Page-TabText);
   )
}
```

Figure A-7. Submodule OS2F0008.C

```
/*============================ Create List Box ================================
FUNCTION:    CreateListBox
DESCRIPTION: Creates a list box (invisible).
PARAMETERS:  Pointer to NOTEBOOK_DATA structure.
             Pointer to PAGE_DATA structure.
RETURNS:     Handle of list box or NULL, if error.
============================================================================*/
#define INCL_WIN

#include <os2.h>
#include "listbox.h"

HWND CreateListBox(HWND hWndClient, USHORT Style, LBSIZE Size, USHORT LbId)
{
   HWND hWndListBox;

   /* Create list box invisible, fill it before showing it. It is more  */
   /* efficient and visually pleasing this way.                         */

   hWndListBox = WinCreateWindow(hWndClient,        /* Parent of window     */
                                 WC_LISTBOX,        /* Class of window      */
                                 "",                /* Window text          */
                                 WS_SYNCPAINT |
                                 Style,              /* Window style        */
                                 Size.x, Size.y,     /* x, y coordinates    */
                                 Size.cx, Size.cy,   /* cx, cy      "       */
                                 hWndClient,         /* Owner               */
                                 HWND_TOP,           /* Behind              */
                                 LbId,               /* List box window ID  */
                                 NULL,               /* Control data        */
                                 NULL);              /* Pres. parameters    */
   return hWndListBox;
}
```

Figure A-8. Submodule OS2F0010.C

```
/*============================ Create Container List ========================
FUNCTION:    CreateContainer
DESCRIPTION: This function creates a container list in the client window.
PARAMETERS:  1. Client window handle.
             2. Container resource identifier.
             3. Position of split bar in pixels.
RETURNS:     Container handle.
===============================================================================*/
#define INCL_PM
#include <os2.h>

HWND CreateContainer(HWND ClienthWnd,
                     USHORT idResource,
                     SHORT sSplitBarPosn,
                     RECTL Coords)
{
   CNRINFO CnrInfo;                    /* Container initialization data    */
   HWND hWndContainer;

   /* Create container window */

   hWndContainer = WinCreateWindow(ClienthWnd,
                                   WC_CONTAINER,
                                   NULL,
                                   WS_VISIBLE | CCS_EXTENDSEL | CCS_READONLY,
                                   Coords.xLeft,
                                   Coords.yBottom,
                                   Coords.xRight,
                                   Coords.yTop,
                                   ClienthWnd,
                                   HWND_TOP,
                                   idResource,
                                   NULL,
                                   NULL);
   CnrInfo.cb            = sizeof(CnrInfo);
   CnrInfo.flWindowAttr  = CV_DETAIL |
                           CA_DETAILSVIEWTITLES |
                           CA_ORDEREDTARGETEMPH;     /* Style            */
   CnrInfo.xVertSplitbar = sSplitBarPosn;

   /* Insert DBCS handling to set CnrInfo.cyLineSpacing, if needed. If    */
   /* DBCS, it will be 8; if SBCS, it will be 4.                          */

   CnrInfo.cyLineSpacing = 4;

   /* Set container details  */

   WinSendMsg(hWndContainer,
              CM_SETCNRINFO,
              MPFROMP(&CnrInfo),
              MPFROMLONG(CMA_FLWINDOWATTR |
                         CMA_XVERTSPLITBAR |
                         CMA_LINESPACING));
   return hWndContainer;
}
```

Figure A-9. Submodule OS2F0011.C

```
/*============================ Set Entry Field Limit ========================
FUNCTION:    SetEntryFieldLimit
DESCRIPTION: Set input limit for given entry field
PARAMETERS:  1. Dialog box handle.
             2. Entry field identifier.
             3. Maximum number of characters allowed.
RETURNS:     Void.
============================================================================*/
#define INCL_PM
#include <os2.h>

VOID SetEntryFieldLimit(HWND hWndDlg, USHORT FieldID, USHORT usLimit)
{
   WinSendDlgItemMsg(hWndDlg,
                     FieldID,
                     EM_SETTEXTLIMIT,
                     MPFROM2SHORT(usLimit,0),
                     NULL);
}
```

Figure A-10. Submodule OS2F0012.C

```
/*============================== Toggle Buttons ==============================
FUNCTION:    ToggleButtons
DESCRIPTION: Toggles a radio/check-box button to a checked or non-checked state
PARAMETERS:  1. HWND   - Window handle.
             2. USHORT - Button identifier.
             3. USHORT - Button state required.
RETURNS:     Void
============================================================================*/
#define INCL_PM
#include <os2.h>

VOID ToggleButtons(HWND hWnd, USHORT CtlID, USHORT CtlState)
{
   WinSendDlgItemMsg(hWnd,
                     CtlID,
                     BM_SETCHECK,
                     MPFROM2SHORT(CtlState, 0),
                     NULL);
   return;
}
```

Figure A-11. Submodule OS2F0013.C

```
/*======================== Check Control Button State ========================
FUNCTION:    IsCtlButtonOn
DESCRIPTION: This function returns the state of a specific radio or check-box
             button.
PARAMETERS:  1. HWND  - Window handle.
             2. USHORT - Control button identifier.
RETURNS:     TRUE if checked, otherwise FALSE.
============================================================================*/
#define INCL_PM
#include <os2.h>

BOOL IsCtlButtonON(HWND hWnd, USHORT CtlID)
{
   return (BOOL)(WinSendDlgItemMsg(hWnd,
                                   CtlID,
                                   BM_QUERYCHECK,
                                   NULL,
                                   NULL));
}
```

Figure A-12. Submodule OS2F0014.C

```
/*======================== Initialize Help Instance ==========================
FUNCTION:    InitHelpInstance
DESCRIPTION: Initialize the help instance.
PARAMETERS:  1. HAB       - Anchor Block handle.
             2. USHORT    - Help Table identifier.
             3. HELPINIT  - Help Instance structure.
             4. PSZ       - Help window title.
             5. PSZ       - Help library name (.HLP file containing all help)
RETURNS:     HWND - Help window (instance) handle.
============================================================================*/
#define INCL_PM
#include <os2.h>
#include <string.h>

HWND InitHelpInstance(HAB hAB,
                      USHORT idHelpTbl,
                      HELPINIT hmiHelpData,
                      PSZ szWinTitle,
                      PSZ szLibName)
{
   HWND  hWndHelpInstance      = (ULONG)NULL; /* Window handle; assume fail*/
   hmiHelpData.cb              = sizeof(HELPINIT);
   hmiHelpData.ulReturnCode    = (ULONG)NULL; /* Store ret. code from init.*/
   hmiHelpData.pszTutorialName = NULL;        /* No tutorial program       */

   /* Indicates help table is defined in the resource file                  */
   hmiHelpData.phtHelpTable    = (PVOID)(0xffff0000 | idHelpTbl);

   /* Action bar not tailored */
   hmiHelpData.hmodAccelActionBarModule = (ULONG)NULL;
   hmiHelpData.idAccelTable             = (ULONG)NULL;
   hmiHelpData.idActionBar              = (ULONG)NULL;
```

Submodule OS2F0014.C (continued)

```
   if (szWinTitle != NULL)                  /* Help window title            */
   {
      strcpy(hmiHelpData.pszHelpWindowTitle, szWinTitle);
   )
   else
   {
      hmiHelpData.pszHelpWindowTitle  = NULL;
      hmiHelpData.hmodHelpTableModule = (ULONG)NULL;/*Help table not in DLL */
      hmiHelpData.fShowPanelId        = (ULONG)NULL;/*Panel ID not displayed*/
   }

   if (szLibName != NULL)                   /* Library containing help panels*/
   {
      strcpy(hmiHelpData.pszHelpLibraryName, szLibName);
   }
   else
   {
      hmiHelpData.pszHelpLibraryName = NULL;
      hWndHelpInstance = WinCreateHelpInstance(hAB, &hmiHelpData);
   }

    if (!hWndHelpInstance)
    {
       WinMessageBox(HWND_DESKTOP, HWND_DESKTOP,
                     (PSZ) "Help Not Available",
                     (PSZ) "Help Creation Error",
                     1,
                     MB_OK | MB_APPLMODAL | MB_MOVEABLE);
    }
    else
    {
       if (hmiHelpData.ulReturnCode)
       {
          WinMessageBox(HWND_DESKTOP,
                        HWND_DESKTOP,
                        (PSZ) "Help Terminated Due to Error",
                        (PSZ) "Help Creation Error",
                        1,
                        MB_OK | MB_APPLMODAL | MB_MOVEABLE);
          WinDestroyHelpInstance(hWndHelpInstance);
          return (HWND)NULL;
       }
    }
    return hWndHelpInstance;
}
```

Figure A-13. Submodule OS2F0015.C

```
/*============================ Insert List Box Item ============================
FUNCTION:     InsertLBItem
DESCRIPTION:  Insert passed string into a list box. Insertion can be in
              ascending or descending sequence, or at the end.
PARAMETERS:   1. HWND - Listbox handle.
              2. INT  - Insertion type:
                        0 = insert in ascending sequence
                        1 = insert in descending sequence
                        2 = insert at end of list (default)
              3. PSZ  - Pointer to string to be inserted.
RETURNS:      SHORT - Index position of item if successful.
                      LIT_MEMERROR if space-allocation error.
                      LIT_ERROR if any other error.
==============================================================================*/
#define INCL_PM
#include <os2.h>
#include <string.h>

SHORT InsertLBItem(HWND hwndListbox, INT Position, PSZ szEntry)
{
   ULONG inspos[] = {LIT_SORTASCENDING,
                     LIT_SORTDECENDING,
                     LIT_END,
                     LIT_END};

   return (SHORT)WinSendMsg(hwndListbox,
                            LM_INSERTITEM,
                            (MPARAM)inpos[ position & 3],
                            (MPARAM)(PSZ)(szEntry));
}
```

Figure A-14. Submodule OS2F0026.C

```
/*======================== Destroy Suballocated Memory =======================
FUNCTION:     UnsetSubMem
DESCRIPTION:  This function ends the use of a memory pool.
REMARKS:      This function releases resources used to manage the suballocation
              of the memory pool. Therefore, this function MUST be called before
              the memory pool is freed.
INPUT:        Base address of memory pool.
RETURNS:      0   - Successful (NO_ERROR)
              532 - Corrupted memory (ERROR_DOSSUB_CORRUPTED)
==============================================================================*/
#define INCL_DOSMEMMGR
#include <os2.h>

USHORT UnsetSubMem(PVOID BaseAddr)
{
   USHORT RtrnCode;
   RtrnCode = DosSubUnsetMem(BaseAddr);
   return RtrnCode;
}
```

Appendix B. Header Files

The following header files are used with the Magic Window application:

DLGBOX.H	Create dialog box	Page 375
HELPIDS.H	Help resource IDs	Page 376
HLPUTILS.H	Help utilities	Page 377
LISTBOX.H	List box	Page 377
MAGICDLG.H	Magic Window dialog box	Page 379
MAGICKW.H	Magic Window header	Page 379
MAGICDP.H	Magic Window dialog box	Page 379
MAGICWP.H	Magic Window header	Page 380
MAGICWP1.H	Magic Window header	Page 380
MENURES.H	Magic Window resource IDs	Page 381
MSGHNDLR.H	Message handler	Page 382
NOTEBOOK.H	Notebook control	Page 383
OS2F.H	OS2F prototypes and declarations	Page 384
PUSHBUTN.H	Push button	Page 385
SYSHELP.H	System help	Page 387
TESTDLG.H	Test dialog	Page 387

The following header files are used with the Warehouse Inventory application:

DBDEFS.H	DB field length definitions	Page 375
DLGBOX.H	Create dialog box	Page 375
GLHELP.H	Global data	Page 375
HELPIDS.H	Help resource IDs	Page 376
HLPUTILS.H	Help utilities	Page 377
LISTBOX.H	List box	Page 377
LLS_DEF.H	Single linked-list structure	Page 378
MSGHNDLR.H	Message handler	Page 382
NOTEBOOK.H	Notebook control	Page 383
OS2F.H	OS2F prototypes and declarations	Page 384
PRNTR.H	Printer	Page 385
RES010.H	Warehouse Inventory resource IDs	Page 385
RES020.H	Warehouse add item IDs	Page 386
RES025.H	Warehouse item IDs	Page 386
RES030.H	Warehouse delete item IDs	Page 386
RES040.H	Warehouse update item IDs	Page 386

SYSHELP.H	System help	Page 387
WINV.H	Warehouse function prototypes	Page 388
WINV010.H	Warehouse mainline structures	Page 390
WINV011.H	Warehouse main window structures	Page 391
WINV020.H	Definitions, variables, & prototypes	Page 392
WINV025.H	Definitions, variables, & prototypes	Page 392
WINV040.H	Definitions, variables, & prototypes	Page 392
WINV050.H	Definitions, variables, & prototypes	Page 392
WINV100.H	Definitions, variables, & prototypes	Page 393
WINV110.H	Definitions, variables, & prototypes	Page 393
WINV120.H	Definitions, variables, & prototypes	Page 393
WINV121.H	Definitions, variables, & prototypes	Page 393
WINV122.H	Definitions, variables, & prototypes	Page 393
WINV125.H	Definitions, variables, & prototypes	Page 394
WINV126.H	Definitions, variables, & prototypes	Page 394
WINV127.H	Definitions, variables, & prototypes	Page 394
WINV128.H	Definitions, variables, & prototypes	Page 394
WINV130.H	Definitions, variables, & prototypes	Page 395
WINV132.H	Definitions, variables, & prototypes	Page 395
WINV133.H	Definitions, variables, & prototypes	Page 395
WINV135.H	Definitions, variables, & prototypes	Page 395
WINV140.H	Definitions, variables, & prototypes	Page 396
WINV148.H	Definitions, variables, & prototypes	Page 396
WINV152.H	Definitions, variables, & prototypes	Page 396
WINV153.H	Definitions, variables, & prototypes	Page 397
WINV154.H	Definitions, variables, & prototypes	Page 397
WINVDEFS.H	Warehouse Inventory definitions	Page 398
WINVMSG.H	Warehouse message definitions	Page 399

Figure B-1. Header File DBDEFS.H

```
/*======================= DB Field Length Definitions ==========================*/
#ifndef _DBDEFS_H
#define _DBDEFS_H

#define DB_ITEM_LGTH 8
#define DB_ITEMDESC_LGTH 45
#define DB_ITEMUNIT_LGTH 5
#define DB_UNITTYPE_LGTH 3
#define DB_WGTUNIT_LGTH 3
#define DB_TOTL_LGTH 6
#define DB_ITEMWT_LGTH 6
#define DB_ITEMQTY_LGTH 6
#define DB_ITEMVALUE_LGTH 7
#define DB_ITEMREORD_LGTH 6
#define DB_TOTLWT_LGTH 6
#define DB_ITEMUPRICE_LGTH 7
#define DB_KEYLEN 20

#endif
```

Figure B-2. Header File DLGBOX.H

```
/*============================ Create Dialog Box ==============================*/
#ifndef _DLGBOX_H
#define _DLGBOX_H

typedef struct _dlgbox_data
{
   HWND    hWndParent;      /* Parent window                                  */
   HWND    hWndOwner;       /* Owner window                                   */
   PFNWP   pfnDlgProc;      /* Dialog procedure to call                       */
   HMODULE hmod;            /* Handle of resource containing dialog template */
   USHORT  idDlg;           /* Dialog resource identity                       */
   PVOID   pCreateParams;   /* Pointer to passed data string                  */
   HWND    hWndHelpInst;    /* Handle to help instance                        */
   HWND    hWndDlg;         /* Handle to dialog                               */
} DLGBOX_DATA;

#endif
```

Figure B-3. Header File GLHELP.H

```
/*================================ Global Data ================================*/
#ifndef _GLHELP_H
#define _GLHELP_H

   EXTERN  HWND    hWndHelpInstance;
   EXTERN  HWND    hWndUser;
   EXTERN  BOOL    fModHelpLoaded;
   EXTERN  HMODULE ModHandleHlp;
   EXTERN  BOOL    fHelpInit = FALSE;
   EXTERN  ULONG   ulResult;
   EXTERN  CHAR    HelpFailBuff[80];

#endif
```

Figure B-4. Header File HELPIDS.H

```
/*============================== Help Resource IDs ==============================*/
#define HLP_BASE_MSG                5000                    /* Base message number */

#define HLP_HELP_NOT_LOADED         5001
#define HLP_HELP_NOT_STARTED        5002
#define HLP_HELP_REQ_FAILED         5003
#define HLP_HELP_TABLEITEM_MISSING  5005
#define HLP_HELP_RUNTIME_ERR        5006
#define HLP_HELP_EXT_PANEL_MISSING  5007

#define IDS_HELP_ERR_TITLE          5100
#define IDS_HELP_WINDOW_TITLE       5101
#define IDS_HELP_FILE_NAME          5102
#define ID_HELPKEYS                 5103
#define IDS_SHOW_PANEL_FLAG         5104
#define IDS_SHOW_UNIT_NAME          5105
#define IDD_ABOUT                   5107
#define IDS_ABOUT_APPNAME           5108
#define IDS_ABOUT_VERSION           5109
#define IDM_ABOUT_BITMAP            5110

#define ID_INFORM_CONTENTS          5111
#define ID_INFORM_INDEX             5112
#define ID_INFORM_KEYS              5113
#define ID_INFORM_EXTENDED          5114
#define ID_INFORM_DISMISS           5115

#define MI_HELP_FOR_HELP            5117
#define MI_HELP_EXTENDED            5118
#define MI_HELP_KEYS                5119
#define MI_HELP_INDEX               5120
#define MI_HELP_ABOUT               5121
#define ID_HELP_KEYS_PANEL          5122
#define ID_HELP_EXTENDED_PANEL      5123

#define ID_HELP_SUBMENU_1           5124
#define ID_HELP_SUBMENU_2           5125

#define ID_HELPTABLE                5130
#define ID_HELPSUBTABLE             5131
#define ID_HELPSUBTABLE1            5132
#define ID_HELPSUBTABLE2            5133
#define ID_HELPSUBTABLE3            5134
#define ID_HELPSUBTABLE4            5135

#define IDD_ABOUT_APPNAME           5139

#define OPT_NBOOK_HELP              5140
#define OPT_MSG_HELP                5141
#define OPT_DLG_HELP                5142
#define OPT_LST_HELP                5143
#define OPT_EXIT_HLP                5144
#define OPT_MGC_HELP                5145
```

Figure B-5. Header File HLPUTILS

```
/*============================== Help Utilities ==============================*/
 PSZ LoadStrResource(ULONG, HWND, HMODULE);
 VOID FreeStrResource(PSZ);
```

Figure B-6. Header File LISTBOX.H

```
/*================================== List Box ================================*/
#ifndef _LISTBOX_H
#define _LISTBOX_H

typedef struct _lbsize
{
    USHORT x;
    USHORT y;
    USHORT cx;
    USHORT cy;
} LBSIZE;

#define IDS_LISTBOX1   9001
#define IDS_LISTBOX2   9002
#define IDS_LISTBOX3   9003
#define IDS_LISTBOX4   9004
#define IDS_LISTBOX5   9005
#define IDS_LISTBOX6   9006
#define IDS_LISTBOX7   9007
#define IDS_LISTBOX8   9008
#define IDS_LISTBOX9   9009
#define IDS_LISTBOX10  9010
#define IDS_LISTBOX11  9011
#define IDS_LISTBOX12  9012
#define IDS_LISTBOX13  9013
#define IDS_LISTBOX14  9014
#define IDS_LISTBOX15  9015
#define IDS_LISTBOX16  9016
#define IDS_LISTBOX17  9017
#define IDS_LISTBOX18  9018
#define IDS_LISTBOX19  9019
#define IDS_LISTBOX20  9020
#define IDS_LISTBOX21  9021
#define IDS_LISTBOX22  9022
#define IDS_LISTBOX23  9023
#define IDS_LISTBOX24  9024
#define IDS_LISTBOX25  9025
#define IDS_LISTBOX26  9026

#endif
```

Figure B-7. Header File LLS_DEF.H

```
/* ======================= Single Linked-List Structure ======================= */
#ifndef __LLS_DEF_H
#define __LLS_DEF_H

typedef struct _LINKTYPE
{
  struct _LINKTYPE  *Next;
  struct _LINKTYPE  *Prev;
  CHAR   *Item;
}LINKTYPE;

typedef struct _LINKLIST
{
  LINKTYPE  *Head;
  LINKTYPE  *Tail;
  LINKTYPE  *CurrNode;                    /* Current node link pointer  */
  ULONG      ItemLgth;
}LINKLIST;

extern LINKLIST  *List;
extern ULONG      BlockOffset;
extern VOID      *BaseAddr;
extern ULONG      ItemSize;

/* Function prototypes    */

VOID        LLSetItemSize(INT);
LINKTYPE *  LLMakeLink(VOID);
USHORT      SubAllocMem(PVOID, PPVOID, ULONG);
USHORT      InitSubMem(PVOID, ULONG);

INT         LLnext(VOID);
VOID        LLHead(VOID);
VOID        LLGetItem(CHAR *);
LINKTYPE *  LLAddNode(CHAR *);
LINKTYPE *  LLAddHeadNode(CHAR *);
VOID        LLInitNode(CHAR *);
VOID        LLDeleteCurrNode(VOID);
INT         LLsetlist(LINKLIST *);
VOID        LLhead(VOID);
VOID        LLtail(VOID);
INT         LLnext(VOID);
INT         LLprev(VOID);

struct WhseRecord * LLFindWhseItem(CHAR *);

#endif
```

Figure B-8. Header File MAGICDLG.H

```
/* ========================= Magic Window Dialog Box ========================== */
#define IDLG_DLGBOX                 4000
#define IDLG_OPNAME                 4020
#define IDLG_MGC_OK                 4030
#define IDLG_MGC_CANCEL             4040
#define IDLG_MGC_HELP               4050
#define IDD_LBOX                    4052
#define IDC_LISTBOX1                4053
#define IDC_COMBOBOX1               4054
#define IDC_ENTRYFLD                4055
```

Figure B-9. Header File MAGICKW.H

```
/ *============================ Magic Window Header ============================ */
#define SEGT_SIZE 70
#define MSG_SIZE 50
#define FLD_SIZE 15
#define SEGT_FLAGS PAG_WRITE | PAG_READ  /* Memory allocation flags */

UCHAR RtrnStr[10];
HAB   hAB;
HWND  hWndMain;
```

Figure B-10. Header File MAGICDP.H

```
/*================================ MAGICDP.H ================================== */

/* ================================ MAGICDP1.C ================================= */
MRESULT EXPENTRY MagicDlgMsgProc(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY DP1InitMsg(HWND);
MRESULT EXPENTRY DP1CmdMsg(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY DP1PaintMsg(HWND);

/* ================================ MAGICDP2.C ================================= */
MRESULT EXPENTRY CtrlDlgMsgProc(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY DP2InitMsg(HWND);
MRESULT EXPENTRY DP2CmdMsg(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY DP2PaintMsg(HWND);
```

Figure B-11. Header File MAGICWP.H

```
/* ============================== Magicwp.H ================================== */
MRESULT EXPENTRY MagickWndProc(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY WMCommandMsg(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY WMControlMsg(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY WMCreateMsg(HWND);
MRESULT EXPENTRY WMDestroyMsg(HWND);
MRESULT EXPENTRY WMMeasureItemMsg(HWND);
MRESULT EXPENTRY WMPaintMsg(HWND);
MRESULT EXPENTRY WMDrawItemMsg(HWND, FONTMETRICS, USHORT, MPARAM);
MRESULT EXPENTRY AboutDlgProc (HWND, ULONG, MPARAM, MPARAM);
```

Figure B-12. Header File MAGICWP1.H

```
/*============================== MAGICWP1.H ====================================
PURPOSE: This file describes the prototypes for the Magic Window procedure. It
         prevent prototypes and declarations from being instantiated more than
         once.
NOTE:    OS/2 datatypes should be declared before this file is included.
==============================================================================*/
#ifndef __MAGICWP1_H
#define __MAGICWP1_H

MRESULT EXPENTRY MagickWndProc(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY WMCommandMsg(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY WMControlMsg(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY WMCreateMsg(HWND);
MRESULT EXPENTRY WMDestroyMsg(HWND);
MRESULT EXPENTRY WMMeasureItemMsg(HWND);
MRESULT EXPENTRY WMPaintMsg(HWND);
MRESULT EXPENTRY WMDrawItemMsg(HWND, FONTMETRICS, ULONG, MPARAM);
MRESULT EXPENTRY AboutDlgProc (HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY LBoxDlgProc (HWND, ULONG, MPARAM, MPARAM);

BOOL FillBoxes(HWND);

#endif
```

Figure B-13. Header File MENURES.H

```
/*======================== Magic Window Resource IDs ========================*/
#define BASE_ID                    1000

#define ID_MAGICK                  1000
#define ID_BOOK                    1005
#define ID_LISTBOX                 1010
#define ID_FRAME_N                 1015
#define ID_FRAME_C                 1020
#define ID_FRAME_L                 1025

#define IDM_OPTIONS                1080
#define IDM_OPTIONS_NOTEBOOK       1085
#define IDM_OPTIONS_DETAILS        1090
#define IDM_OPTIONS_LISTBOX        1095
#define IDM_OPTIONS_CONTROLS       1100
#define IDM_OPTIONS_MSGBOX         1105
#define IDM_OPTIONS_DLGBOX         1110
#define IDM_OPTIONS_EXIT_F3        1115

#define IDM_ABOUT                  2005
#define IDB_ABOUT_BITMAP           2006

#define IDM_HELP                   1170
#define IDM_HELP_FORHELP           1171
#define IDM_HELP_XTENDED           1172
#define IDM_HELP_KEYS              1173
#define IDM_HELP_ABOUT             1174

#define IDM_FILE                   3000
#define IDM_FILE_OPEN              3001
#define IDM_FILE_SAVE              3002
#define FILE_OPEN                  3003
#define FILE_SAVE                  3004

#define IDC_CONTAINER              1180

#define IDS_ERR_WINDOW_CREATE      1200
#define IDS_ERR_WINDOW_POS         1210
#define IDS_MENU_TITLE             1215

#define IDS_USR_MSG1               1220
#define IDS_USR_MSG2               1225
#define IDS_USR_MSG3               1230
```

Figure B-14. Header File MSGHNDLR.H

```
/* ============================ Message Handler ================================
DESCRIPTION: This header file contains the definitions for the pop-up message
             display functions.
REMARKS:     1. Include MSGHNDLR.LIB in the application makefile link command.
             2. Include MSGHNDLR.RC1 in the application resource file (rcinclude
                MSGHNDLR.RC1)
============================================================================== */

/* ========================= Message Class Definitions ======================== */
#define ERR_CLASS1_CRITICAL       0000
#define ERR_CLASS2_ERROR          0001
#define ERR_CLASS3_WARNING        0002
#define ERR_CLASS4_INFORM         0003

/* ========================= Message Class Resource IDs ======================== */
#define IDS_MSGCLASS_CRITICAL          7001
#define IDS_MSGCLASS_ERROR             7002
#define IDS_MSGCLASS_WARNING           7003
#define IDS_MSGCLASS_INFO              7004

/* ============================ Defined Constants ============================== */
#ifndef BUFF_SIZE
   #define BUFF_SIZE   500
#endif

#ifndef TRUE
   #define TRUE          1
#endif

#ifndef FALSE
   #define FALSE         0
#endif

#ifndef INSERT
   #define INSERT       '%'
#endif

#ifndef TAB
   #define TAB         '\t'
#endif

#ifndef INSRT_FLAG
   #define INSRT_FLAG   '%'
#endif

#ifndef MISMATCH
   #define MISMATCH      2
#endif

#ifndef MEM_ERR
   #define MEM_ERR       3
#endif

USHORT DisplayMessage(ULONG, HWND, USHORT, CHAR *);
INT    InsertSubStrings(PSZ, PSZ);
```

Figure B-15. Header File NOTEBOOK.H

```
/*============================ Notebook Control ==============================*/
#ifndef _NOTEBOOK_H
#define _NOTEBOOK_H

#define MAX_PAGES 50

typedef struct _PageData
{
   ULONG  ulPageId;
   USHORT usPageAttrib;
   USHORT usPgPosn;
   CHAR   *StatusText;
   CHAR   *TabText;
} PAGE_DATA;

typedef struct _NoteBookData
{
   HWND    hWndNotebook;            /* Notebook handle          */
   USHORT  NotebookID;              /* Notebook id              */
   USHORT  sxMajorTab;              /* Major Tab sizes          */
   USHORT  syMajorTab;
   USHORT  usMajorPgAttr;           /* Major page attribute     */
   USHORT  sxMinorTab;              /* Minor Tab sizes          */
   USHORT  syMinorTab;
   USHORT  usMinorPgAttr;           /* Minor page attribute     */
   USHORT  xLeft;                   /* Notebook size            */
   USHORT  xRight;
   USHORT  yBottom;
   USHORT  yTop;
   ULONG   ulNotebookStyle;         /* Notebook style           */
   PAGE_DATA *NBPages[MAX_PAGES];
} NOTEBOOK_DATA;

#endif
```

Figure B-16. Header File OS2F.H

```
/* ================================ OS2F.H ===================================
PURPOSE: This file contains the prototypes and declarations required to allow
         access to the functions in the OS2FXXXX.C files
NOTE:    OS/2 datatypes must be declared prior to including this file in user
         application code
============================================================================== */
#ifndef __OS2F_H
#define __OS2F_H

#ifndef __NOTEBOOK_H
   #include "notebook.h"
#endif

#ifndef __DLGBOX_H
   #include "dlgbox.h"
#endif

#ifndef __LISTBOX_H
   #include "listbox.h"
#endif

/* ================================ OS2F0001.C ================================= */
LONG RtJustCharString (HPS, POINTL *, LONG, PSZ);

/* ================================ OS2F0002.C ================================= */
VOID DisplayDBerror(LONG);

/* ================================ OS2F0003.C ================================= */
USHORT GetSelectedLBEntry(HWND);

/* ================================ OS2F0004.C ================================= */
HWND CreateWindow(HWND, ULONG, CHAR*, CHAR*, ULONG, INT, INT, INT, INT, PHWND,
                  ULONG, ULONG);

/* ================================ OS2F0006.C ================================= */
USHORT CreateWinNoteBook(HWND, NOTEBOOK_DATA *);

/* ================================ OS2F0007.C ================================= */
VOID InsertNotebookPage(NOTEBOOK_DATA *, PAGE_DATA *);

/* ================================ OS2F0008.C ================================= */
HWND CreateListBox(HWND, USHORT, LBSIZE, USHORT);

/* ================================ OS2F0010.C ================================= */
HWND CreateContainer(HWND, USHORT, SHORT, RECTL);

/* ================================ OS2F0011.C ================================= */
VOID SetEntryFieldLimit(HWND, USHORT, USHORT);

/* ================================ OS2F0012.C ================================= */
VOID ToggleButtons(HWND, USHORT, USHORT);

/* ================================ OS2F0013.C ================================= */
BOOL IsCtlButtonON(HWND, USHORT);

/* ================================ OS2F0014.C ================================= */
HWND InitHelpInstance(HAB, USHORT, HELPINIT, PSZ, PSZ);

/* ================================ OS2F0015.C ================================= */
SHORT InsertLBItem(HWND, INT, PSZ);

#endif
```

Figure B-17. Header File PRNTR.H

```
/*================================ Printer ==================================*/
#ifdef PRNTR
   APIRET PrntOK(USHORT *);
   APIRET PrntInit(VOID);
   APIRET PrntrPutChar(INT);
   APIRET PrntrPutStr(CHAR *);
   APIRET PrntAtPos(CHAR *, INT Row, INT Col);
   APIRET PrntNewPage(BOOL);
   APIRET OpenPrinter(VOID);
#else
   EXTERN APIRET PrntOK(USHORT *);
   EXTERN APIRET PrntInit(VOID);
   EXTERN APIRET PrntrPutChar(INT);
   EXTERN APIRET PrntrPutStr(CHAR *);
   EXTERN APIRET PrntAtPos(CHAR *, INT Row, INT Col);
   EXTERN APIRET PrntNewPage(BOOL);
   EXTERN APIRET OpenPrinter(VOID);
#endif
```

Figure B-18. Header File PUSHBUTN.H

```
/*============================== Push Button ================================*/
#define IDLG_OK                      103
#define IDLG_CANCEL                  104
#define IDLG_HELP                    105
```

Figure B-19. Header File RES010.H

```
/*====================== Warehouse Inventory Resource IDs ====================
DESCRIPTION: Contains resource ids required for Warehouse Inventory application
REMARKS:     Include header file with WINV010.C and WINV010.RC
============================================================================*/
#define BASE_ID                      1000
#define ID_WINV010                   1000
#define ID_WINVLIST                  1010
#define IDM_OPTIONS                  1020
#define IDM_OPTIONS_ADD_ITEM         1030
#define IDM_OPTIONS_DELETE_ITEM      1040
#define IDM_OPTIONS_EXIT_F3          1060
#define IDM_OPTIONS_EXIT             1065
#define IDM_OPTIONS_UPDT_ITEM        1070
#define IDM_HELP                     1090
#define IDM_HELP_FORHELP             1100
#define IDM_HELP_XTENDED             1110
#define IDM_HELP_KEYS                1120
#define IDM_ABOUT                    1130

#define IDS_ERR_WINDOW_CREATE        1300
#define IDS_ERR_WINDOW_POS           1305
#define IDS_WHSE_MENU_TITLE          1315
#define IDS_WHSE_MAIN_HDG            1320
#define IDS_ADD_TITLE                1325
#define IDS_DEL_TITLE                1330
#define IDS_ADD_INBR                 1335
#define IDS_ADD_IDESC                1340
#define IDS_ADD_IUNIT                1345
#define IDS_ADD_IPRICE               1350
#define IDS_ADD_IQTY                 1355
```

Figure B-20. Header File RES020.H

```
/*========================== Warehouse Add Item IDs ============================*/
#define IDLG_ADD_020                1500
#define IDLG_ADD_PPU                1503
#define IDLG_ADD_DESC               1507
#define IDLG_ADD_WT                 1508
#define IDLG_ADD_WTUNIT             1509
#define IDLG_ADD_QTY                1510
#define IDLG_ADD_REORD              1512
#define IDLG_ADD_PKG_UNIT           1513
#define IDLG_ADD_ITEM               1522

#define ADD_HELP                    1521
#define ADD_CANCEL                  1520
#define ADD_OK                      1519
```

Figure B-21. Header File RES025.H

```
/*========================== Warehouse Item IDs ================================*/
#define IDLG_ABOUT            4000
#define IDLG_OP               4010
#define IDLG_OPNAME           4020
#define IDLG_ABOUT_OK         4030
#define IDLG_ABOUT_CANCEL     4040
#define IDLG_ABOUT_HELP       4050
```

Figure B-22. Header File RES030.H

```
/*========================== Warehouse Delete Item IDs ==========================*/
#define IDLG_DEL_ITEM          2000
#define IDLG_DEL_DESC          2010
#define IDLG_DEL_OK            2015
#define IDLG_DEL_030           2020
#define IDLG_DEL_CANCEL        2025
#define DEL_HELP               2030
```

Figure B-23. Header File RES040.H

```
/*========================== Warehouse Update Item IDs ==========================*/
#define IDLG_UPD_OK              519
#define IDLG_UPD_CANCEL          520
#define IDLG_UPD_HELP            521
#define IDLG_UPD_040             500
#define IDLG_UPD_PRICE           503
#define IDLG_UPD_DESC            507
#define IDLG_UPD_WGT             508
#define IDLG_UPD_WT_UNIT         509
#define IDLG_UPD_QTY             510
#define IDLG_UPD_REORD           512
#define IDLG_UPD_PKUNIT          513
#define IDLG_UPD_ITEM            522
```

Figure B-24. Header File SYSHELP.H

```
/*============================== System Help ==============================*/
#ifndef _SYSHELP_H
#define _SYSHELP_H

#ifndef BUFF_SIZE
   #define BUFF_SIZE 256
#endif

BOOL    EXPENTRY    HlpOpenHelp(HWND, HMODULE);
BOOL    EXPENTRY    HlpAssociateHelp(HWND);
BOOL    EXPENTRY    HlpDestroyHelp (VOID);
BOOL    EXPENTRY    HlpCaseHM_Messages(ULONG, MPARAM, ULONG*);
BOOL    EXPENTRY    HlpCaseWM_Command(ULONG);
BOOL    EXPENTRY    HlpDisAssociateHelp(HWND);

MRESULT EXPENTRY    HlpSendMsg(ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY    HlpAboutDlgProc(HWND, ULONG, MPARAM, MPARAM);

PSZ LoadStrResource(ULONG, HWND, HMODULE);
VOID FreeStrResource(PSZ);

HWND hWndHelpInstance;       /* Handle to a help instance */

#endif
```

Figure B-25. Header File TESTDLG.H

```
/*============================== Test Dialog ==============================*/
#define IDLG_DISPLAY                200
#define IDLG_ENTRY1                 201

#define IDLG_CHK1                   202
#define IDLG_CHK2                   203

#define IDLG_RAD1                   204
#define IDLG_RAD2                   205
#define IDLG_RADGRP                 206

#define IDLG_COMBO                  207
#define IDLG_SPIN                   208
#define IDLG_SLIDER                 209

#define IDLG_TST_OK                 210

#define IDLG_TST_CANCEL             211
#define IDLG_TST_HELP               212
```

Figure B-26. Header File WINV.H

```
/* ================================== WINV.H ================================== */
#ifndef __WINV_H
#define __WINV_H

#ifndef __WINV010_H
   #include "WINV010.h"
#endif

#ifndef __WINV011_H
   #include "WINV011.h"
#endif

#ifndef __WINV020_H
   #include "WINV020.h"
#endif

#ifndef __WINV025_H
   #include "WINV025.h"
#endif

#ifndef __WINV040_H
   #include "WINV040.h"
#endif

#ifndef __WINV050_H
   #include "WINV050.h"
#endif

#ifndef __WINV100_H
   #include "WINV100.h"
#endif

#ifndef __WINV110_H
   #include "WINV110.h"
#endif

#ifndef __WINV120_H
   #include "WINV120.h"
#endif

#ifndef __WINV121_H
   #include "WINV121.h"
#endif

#ifndef __WINV122_H
   #include "WINV122.h"
#endif

#ifndef __WINV125_H
   #include "WINV125.h"
#endif

#ifndef __WINV126_H
   #include "WINV126.h"
#endif

#ifndef __WINV127_H
   #include "WINV127.h"
#endif
```

Header File WINV.H (continued)

```
#ifndef __WINV128_H
   #include "WINV128.h"
#endif

#ifndef __WINV130_H
   #include "WINV130.h"
#endif

#ifndef __WINV132_H
   #include "WINV132.h"
#endif

#ifndef __WINV133_H
   #include "WINV133.h"
#endif

#ifndef __WINV135_H
   #include "WINV135.h"
#endif

#ifndef __WINV140_H
   #include "WINV140.h"
#endif

#ifndef __WINV148_H
   #include "WINV148.h"
#endif

#ifndef __WINV152_H
   #include "WINV152.h"
#endif

#ifndef __WINV153_H
   #include "WINV153.h"
#endif

#ifndef __WINV154_H
   #include "WINV154.h"
#endif

#endif
```

Figure B-27. Header File WINV010.H

```
/*==================== Warehouse Inventory Mainline Structure ====================
DESCRIPTION: Contains all definitions, variables, and function prototypes
             required by the Warehouse Inventory mainline.
REMARKS:     Include header file with WHINV010.C
================================================================================*/
#ifndef __WINV010_H
#define __WINV010_H

#ifndef __LLS_DEF_H
   #include "lls_def.h"
#endif

#ifndef __WINVDEFS_H
   #include "winvdefs.h"
#endif

typedef struct _dbase
{
   PVOID     BaseAddr;
   LINKLIST  WhouseList;
   LINKTYPE *NodePtr;
} DBASE, *DBASE_MEM;

/* These structures are used to add the program name to the task list.        */

EXTERN HSWITCH hSwitch;
EXTERN SWCNTRL pSwctl;
EXTERN CHAR    szAppName[20];              /* Class name of application       */

EXTERN WHSE_RECORD  WhseRecord;
EXTERN WHSE_PTR     WhsePtr;

EXTERN WHSE_RECORD  xItemRecord;
EXTERN WHSE_PTR     pWhse;

#endif
```

Figure B-28. Header File WINV011.H

```
/*============================== WINV011.H ======================================
DESCRIPTION: Contains all definitions, variables, and function prototypes
             required by the Warehouse Inventory main window procedure.
================================================================================*/
#ifndef __WINV011_H
#define __WINV011_H

#ifndef __WINV010_H
   #include "winv010.h"
#endif

#ifndef __WINVDEFS_H
   #include "winvdefs.h"
#endif

typedef struct _params
{
   HWND     hWndList;
   WHSE_PTR pWhseRec;
}PARAMS, *PPARAMS;

EXTERN DBASE       WinvDB;
EXTERN DBASE_MEM   WinvDBPtr;

EXTERN FONTMETRICS systemfont;

EXTERN CHAR szAppName[20];   /* Class name of application                    */

EXTERN HAB  hAB;             /* Handle to the Anchor Block                   */
EXTERN HMQ  hMQ;             /* Handle to the Message Queue                  */
EXTERN HWND hWndFrame;       /* Handle to the Main Window Frame              */
EXTERN HWND hWndClient;      /* Handle to the Client Window                  */
EXTERN HWND hWndInitMsg;     /* Handle to the initial msg. dialog box        */
EXTERN HWND hWndListBox;     /* Handle to Warehouse Inventory List Box       */
EXTERN HWND hWndDetail;      /* Handle to "Item Detail" Dialog               */

/*=========================== Function Prototypes ============================*/
MRESULT EXPENTRY WINVDemoWndProc(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY WMCommandMsg(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY WMCreateMsg(HWND);
MRESULT EXPENTRY WMControlMsg(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY WMPaintMsg(HWND);
MRESULT EXPENTRY WMDrawItemMsg(HWND, FONTMETRICS, MPARAM);
MRESULT EXPENTRY WMMeasureItemMsg(HWND);

#endif
```

Figure B-29. Header File WINV020.H

```
/* ================================ WINV020.H ================================= */
#ifndef __WINV020_H
#define __WINV020_H

MRESULT EXPENTRY WhInvAddItem(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY WMCreateMsg020(HWND, MPARAM);
MRESULT EXPENTRY WMCommandMsg020(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY WMControlMsg020(HWND, ULONG, MPARAM);

#endif
```

Figure B-30. Header File WINV025.H

```
/* ================================ WINV025.H ================================= */
#ifndef __WINV025_H
#define __WINV025_H

MRESULT EXPENTRY AboutDlgProc(HWND, ULONG, MPARAM, MPARAM);

#endif
```

Figure B-31. Header File WINV040.H

```
/* ================================ WINV040.H ================================= */
#ifndef __WINV040_H
#define __WINV040_H

#ifndef __WINV011_H
   #include "winv011.h"
#endif

EXTERN PPARAMS     ppItem;
EXTERN PARAMS      pItem;

MRESULT EXPENTRY WhInvUpdateItem(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY WMCreateMsg040(HWND, MPARAM);
MRESULT EXPENTRY WMCommandMsg040(HWND, ULONG, MPARAM, MPARAM);
MRESULT EXPENTRY WMControlMsg040(HWND, ULONG, MPARAM);

#endif
```

Figure B-32. Header File WINV050.H

```
/* ================================ WINV050.H ================================= */
#ifndef __WINV050_H
#define __WINV050_h

ULONG WhInvUtilIO(VOID);

#endif
```

Figure B-33. Header File WINV100.H

```
/* ================================ WINV100.H ================================= */
#ifndef __WINV100_H
#define __WINV100_H

USHORT GetSelectedLine(HWND);

#endif
```

Figure B-34. Header File WINV110.H

```
/* ================================ WINV110.H ================================= */
#ifndef __WINV110_H
#define __WINV110_H

PSZ GetSelectedItem(HWND, USHORT);

#endif
```

Figure B-35. Header File WINV120.H

```
/* ================================ WINV120.H ================================= */
#ifndef __WINV120_H
#define __WINV120_H

#ifndef __WINVDEFS_H
   #include "winvdefs.h"
#endif

VOID ClearWhseRecord(WHSE_PTR);

#endif
```

Figure B-36. Header File WINV121.H

```
/* ================================ WINV121.H ================================= */
#ifndef __WINV121_H
#define __WINV121_H

VOID SetAddFieldLgths(HWND);

#endif
```

Figure B-37. Header File WINV122.H

```
/* ================================ WINV122.H ================================= */
#ifndef __WINV122_H
#define __WINV122_H

#ifndef __WINVDEFS_H
   #include "windefs.h"
#endif

VOID GetAddDlgEntries(USHORT, HWND, ITEM_PTR);

#endif
```

Figure B-38. Header File WINV125.H

```
/* ================================ WINV125.H ================================ */
#ifndef __WINV125_H
#define __WINV125_H

#ifndef __WINVDEFS_H
   #include "windefs.h"
#endif

PSZ BuildWhseInvLine(WHSE_PTR);

#endif
```

Figure B-39. Header File WINV126.H

```
/* ================================ WINV126.H ================================ */
#ifndef __WINV126_H
#define __WINV126_H

#ifndef __WINVDEFS_H
   #include "windefs.h"
#endif

INT AddInvItem(HWND, ITEM_PTR);

#endif
```

Figure B-40. Header File WINV127.H

```
/* ================================ WINV127.H ================================ */
#ifndef __WINV127_H
#define __WINV127_H

#ifndef __WINVDEFS_H
   #include "winvdefs.h"
#endif

INT DeleteInventoryItem(HWND, USHORT, WHSE_PTR);

#endif
```

Figure B-41. Header File WINV128.H

```
/* ================================ WINV128.H ================================ */
#ifndef __WINV128_H
#define __WINV128_H

#ifndef __WINVDEFS_H
#include "windefs.h"
#endif

INT UpdateInvItem(HWND, WHSE_PTR);

#endif
```

Figure B-42. Header File WINV130.H

```
/* ================================ WINV130.H ================================ */
#ifndef __WINV130_H
#define __WINV130_H

#ifndef __WINV010_H
   #include "winv010.h"
#endif

EXTERN DBASE_MEM MemPtr;
EXTERN PSZ WhseDisplyLine;
EXTERN INT ListWhseInventory(HWND);
INT ListWhseInventory(HWND);

#endif
```

Figure B-43. Header File WINV132.H

```
/* ================================ WINV132.H ================================ */
#ifndef __WINV132_H
#define __WINV132_H

#ifndef __WINVDEFS_H
   #include "windefs.h"
#endif

VOID ClearItemRecord(ITEM_PTR);

#endif
```

Figure B-44. Header File WINV133.H

```
/* ================================ WINV133.H ================================ */
#ifndef __WINV133_H
#define __WINV133_H

VOID StopDB(PVOID);
#endif
```

Figure B-45. Header File WINV135.H

```
/*================================ WINV135.H ================================*/
#ifndef __WINV135_H
#define __WINV135_H

#ifndef __WINV010_H
   #include "winv010.h"
#endif

#define LISTMAX 4000
EXTERN DBASE_MEM MemPtr;
EXTERN ULONG     BlockOffset;
EXTERN VOID     *BaseAddr;

#ifndef BUFF_SIZE
   #define BUFF_SIZE 100
#endif

USHORT StartDB(DBASE *);
#endif
```

Figure B-46. Header File WINV140.H

```
/* ================================ WINV140.H ================================ */
#ifndef __WINV140_H
#define __WINV140_H

#ifndef __WINVDEFS_H
   #include "windefs.h"
#endif

VOID GetUpdDlgEntries(ULONG, HWND, ITEM_PTR);

#endif
```

Figure B-47. Header File WINV148.H

```
/* ================================ WINV148.H ================================ */
#ifndef __WINV148_H
#define __WINV148_H

#ifndef __WINVDEFS_H
   #include "windefs.h"
#endif

INT LoadWhseDBRecord(WHSE_PTR);

#endif
```

Figure B-48. Header File WINV152.H

```
/* ================================ WINV152.H ================================ */
#ifndef __WINV152_H
#define __WINV152_H

#ifndef __WINV010_H
   #include "winv010.h"
#endif

#ifndef __WINVDEFS_H
   #include "winvdefs.h"
#endif

#define LISTMAX 4000

extern DBASE_MEM MemPtr;

INT AddDBRec(WHSE_PTR);

#endif
```

Figure B-49. Header File WINV153.H

```
/* ========================== WINV153.H ======================================= */
#ifndef __WINV153_H
#define __WINV153_H

#ifndef __WINV010_H
#include "winv010.h"
#endif

#ifndef __WINVDEFS_H
   #include "winvdefs.h"
#endif

EXTERN DBASE_MEM MemPtr;

INT UpdDBRec(WHSE_PTR);

#endif
```

Figure B-50. Header File WINV154.H

```
/* ============================== WINV154.H ================================== */
#ifndef __WINV154_H
#define __WINV154_H

#ifndef __WINVDEFS_H
   #include "winvdefs.h"
#endif

INT DelDBRec(WHSE_PTR);

#endif
```

Figure B-51. Header File WINVDEFS.H

```
/*====================== Warehouse Inventory Definitions ========================*/
#ifndef __WINVDEFS_H
#define __WINVDEFS_H

#ifndef _DBDEFS_H
#include "dbdefs.h"
#endif

/*=========================== DB Access Return Codes ============================ */
#define OK              0
#define ERROR          -1
#define DUPLICATE      -2
#define MISMATCH       -4
#define NOTFOUND       -6
#define TBL_EMPTY      -8
#define CANCELLED      -10
#define NO_ITEM_KEY    -12
#define NO_QTY         -14
#define NO_WGT         -15
#define REREAD_ERROR   -16
#define NO_WGT_UNIT    -17
#define EDIT_ERROR     -18
#define NO_REORD       -19
#define NOT_EMPTY      -20
#define DB_NOTSTARTED -22
#define STILL_STOCKED -24
#define DB_OVERLD      -25

/*========================= Warehouse Inventory Record ========================*/
typedef struct WhseRecord
{
   CHAR szWhouseKey[DB_KEYLEN];
   CHAR szWhouseID[DB_ITEM_LGTH + 1];                    /* Item ID              */
   CHAR szWhouseDesc[DB_ITEMDESC_LGTH + 1];              /* Item description     */
   CHAR szWhouseQOH[DB_ITEMQTY_LGTH + 1];                /* Total item quantity */
   CHAR szWhouseUnit[DB_ITEMUNIT_LGTH + 1];              /* Item unit code       */
   CHAR szWhTotlWt[DB_TOTLWT_LGTH + 1];                  /* Total item weight    */
   CHAR szWhWgtUnit[DB_WGTUNIT_LGTH + 1];                /* Weight unit          */
   CHAR szWhouseReord[DB_ITEMREORD_LGTH + 1];            /* Item reorder point   */
   CHAR szWhouseUnitPrice[DB_ITEMUPRICE_LGTH + 1];       /* Item unit price      */
} WHSE_RECORD, *WHSE_PTR;

#define WHSE_RECLEN sizeof(WHSE_RECORD)
#define WHSE_NODE   WHSE_RECLEN
#define WHSE_LIST   WHSE_RECLEN * 200                    /* Arbitrary length     */

#endif
```

Figure B-52. Header File WINVMSG.H

```
/* ================================ WINVMSG.H ================================
DESCRIPTION: Contains all definitions for use with the OS/2 message system used
             by the Warehouse Inventory application.
============================================================================== */

#define   IDS_WHINV_MSG_01              6001
#define   IDS_WHINV_MSG_02              6002
#define   IDS_WHINV_MSG_03              6003
#define   IDS_WHINV_MSG_04              6004
#define   IDS_WHINV_MSG_05              6005
#define   IDS_WHINV_MSG_06              6006
#define   IDS_WHINV_MSG_07              6007
#define   IDS_WHINV_MSG_08              6008
#define   IDS_WHINV_MSG_09              6009
#define   IDS_WHINV_MSG_10              6010
#define   IDS_WHINV_MSG_11              6011
#define   IDS_WHINV_MSG_12              6012
#define   IDS_WHINV_MSG_13              6013
#define   IDS_WHINV_MSG_14              6014
#define   IDS_WHINV_MSG_15              6015
#define   IDS_WHINV_MSG_16              6016
#define   IDS_WHINV_MSG_17              6017
#define   IDS_WHINV_MSG_18              6018
#define   IDS_WHINV_MSG_19              6019

#define   IDS_WHINV_MSG_20              6020
#define   IDS_WHINV_MSG_22              6022
#define   IDS_WHINV_MSG_24              6024
#define   IDS_WHINV_MSG_25              6025
#define   IDS_WHINV_MSG_26              6026
#define   IDS_WHINV_MSG_27              6027
#define   IDS_WHINV_MSG_28              6028
#define   IDS_WHINV_MSG_29              6029

#define   IDS_WHINV_MSG_30              6030
#define   IDS_WHINV_MSG_31              6031
#define   IDS_WHINV_MSG_32              6032
#define   IDS_WHINV_MSG_34              6034
#define   IDS_WHINV_MSG_35              6035
#define   IDS_WHINV_MSG_36              6036
#define   IDS_WHINV_MSG_37              6037
#define   IDS_WHINV_MSG_38              6038
#define   IDS_WHINV_MSG_39              6039

#define   IDS_WHINV_MSG_40              6040
#define   IDS_WHINV_MSG_41              6041
#define   IDS_WHINV_MSG_42              6042
#define   IDS_WHINV_MSG_43              6043
#define   IDS_WHINV_MSG_44              6044
#define   IDS_WHINV_MSG_45              6045
#define   IDS_WHINV_MSG_46              6046
#define   IDS_WHINV_MSG_47              6047
#define   IDS_WHINV_MSG_48              6048
#define   IDS_WHINV_MSG_49              6049

#define   IDS_WHINV_MSG_50              6050
#define   IDS_WHINV_MSG_51              6051
#define   IDS_WHINV_MSG_52              6052
#define   IDS_WHINV_MSG_53              6053
#define   IDS_WHINV_MSG_54              6054
#define   IDS_WHINV_MSG_55              6055
```

Header File WINVMSG.H (continued)

```
#define   IDS_WHINV_MSG_56              6056
#define   IDS_WHINV_MSG_57              6057
#define   IDS_WHINV_MSG_58              6058
#define   IDS_WHINV_MSG_59              6059

#define   IDS_WHINV_MSG_60              6060
#define   IDS_WHINV_MSG_61              6061
#define   IDS_WHINV_MSG_62              6062
```

Appendix C.
More Tips, Time-Savers, and Good Ideas

Design Tips

1. For the least work and the best programs, *plan your application before beginning to code*!
2. Choose a main purpose for your application and keep it in the forefront of your design plan. Base this purpose on the needs and wants of your users. Don't guess -- ask them!
3. Build your application's features around the main purpose. This will eliminate coding in unnecessary functions that might obscure your program's ease of use and overcomplicate the learning process of your users.
4. Plan the basic features of your application based on:
   a. Memory requirements
   b. Input (data communication)
   c. Processing methods
   d. Output (displaying and printing)
   e. Interfacing (sharing data among other applications)
   f. Saving data
   g. Protecting data
5. Consider the best way to display your features, using the following guidelines:
   a. Make sure the visual presentation is clean and easy to read.
   b. Be consistent in spelling and placement of repetitive title names, field entries, etc.
   c. Create an outline of the actions performed by your users to accomplish the purpose of your application. Structure the presentation of each task in a logical sequence that follows that outline.
   d. Test your structure on some beta users and listen to their feedback.
6. Make your feature easy to use by taking advantage of the Workplace Shell. Use menu choices, submenus, dialog boxes, pop-up windows, drag and drop, and hot keys (see "Accelerator Keys and Shortcuts" below), wherever appropriate, to provide your users with as much flexibility as possible.

7. Remember to keep your design portable and easy to maintain so future upgrades to your application will be problem-free.

Setting Passwords and Screen Savers

The OS/2 operating system provides a tab called *Lockup* on the Settings notebook of the Desktop that enables you to lock your keyboard and mouse, select a screen saver, and set a password. The Lockup tab consists of three pages:

- Page 1 enables you to set an automatic lockup and select the timeout value.
- Page 2 enables you to select and size a screen saver. Ensure that the Auto-dim box is checked. This will cause your screen saver to gradually fade to black and display a moving padlock across the screen.
- Page 3 enables you to create a password for locking your system. To re-enter your system, simply type your password and press Enter.

Accelerator Keys and Shortcuts

The OS/2 operating system allows you to use a combination of function keys or *accelerators* to speed up your keyboard entry. Some of the most common accelerator keys and other shortcuts are shown below. Additional information on key combinations can be retrieved from the **Keys Help** option of any Help window.

Accelerator Keys

Alt+Backspace	Undoes a previous action.
Alt+Esc	Switches to the next icon, session, or open window.
Alt+F4	Closes an active window.
Alt+F5	Restores a window to its previous position and size.
Alt+F6	Toggles the cursor between two windows.
Alt+F9	Minimizes a window.
Alt+F10	Maximizes a window.
Alt+Home	Switches between a DOS window and DOS full screen.
Alt+Ins	Copies an object.
Alt+Shift+Tab	Makes the Desktop active.

Alt+Tab	Switches to the next window.
Alt+underlined letter	Selects an item from a drop-down menu.
Ctrl+/	Selects all items in a list or all objects in a window.
Ctrl+	Deselects all objects.
Ctrl+Esc	Displays the Window List.
Ctrl+left mouse button	Selects multiple objects.
Ctrl+right mouse button	Drags an object to be copied.
Shift+Alt+Backspace	Undoes the previous Undo.
Shift+Del	Cuts an object from the selected area.
Shift+Esc	Displays a window menu.
Shift+F10	Displays pop-up menu for an object or the Desktop.
Shift+Ins	Pastes an object into the selected area.
Shift+right mouse button	Moves object from Templates folder to the Desktop.
Shift+Tab	Moves backwards in a dialog box.

Other Shortcuts

ESC	Removes a displayed menu or active window.
F1	Displays Help windows or information.
F5	Refreshes a window.
F10	Switches to the menu bar of an active window.
F11	Displays the Help Index.
Tab	Moves forward in a dialog box.

Error Message Solutions

- ☐ **20059 - SOMERROR_DefaultMethod**

  Ensure that the method has been defined before invoking it.

- ☐ **20069 - SOMERROR_MissingMethod**

  Ensure that the method is defined on the target object.

☐ **20089 - SOMERROR_NullId**

Check for null IDs.

☐ **20169 - SOMERROR_BadOverride**

Ensure that the method to be overridden is defined in its parent class.

☐ **20179 - SOMERROR_NotImplementedYet**

Ensure that the method is properly implemented.

☐ **20199 - SOMERROR_BadArgument**

Ensure that the argument is valid.

☐ **SYS0002E or SYS0003E - System cannot find the file or path specified.**

Ensure that the correct file/path is specified and retry command.

☐ **SYS0005E - Access denied.**

The following solutions depend on the reason access was denied:

- Ensure that the correct filename is specified.
- If extended attributes have been lost, use the CHKDSK command with the /F option to restore them.
- If a read-only file, use the ATTRIB command to change it.
- If the resource is in use, wait until it is no longer active.

☐ **SYS0015E - System cannot find drive specified.**

Ensure that the correct drive letter is used. Note that the RESTORE command cannot be used on a read-only or redirect drive.

☐ **SYS0028E - Printer is out of paper or insufficient space to create spooled file.**

Ensure that printer is properly installed and turned on; then check that paper is loaded correctly. If disk space is insufficient, remove any unnecessary files and try again.

☐ **SYS0029E or SYS0030E - System cannot write to, or read from, the device specified.**

Ensure that the device is properly installed and turned on. It should be in the correct mode (send or receive) and not currently in use by another process.

☐ **SYS0032E or SYS0033E - Process cannot access file because it is being used by another process.**

Retry the command later.

☐ **SYS0035E - Program could not open file.**

Increase the value in the FCBS statement in the CONFIG.SYS file, reboot, and retry the command.

☐ **SYS0082E - Directory or file cannot be created.**

Ensure that the file or directory does not already exist and that the entire path is correctly specified. If the disk or directory is full, remove unnecessary files and try again.

☐ **SYS0126E - Module cannot be found.**

Ensure that the correct module is being requested and is loaded.

☐ **SYS0127E - Procedure cannot be found.**

Ensure that the correct procedure is being requested and that it exists in the module or Exitlist routine list.

☐ **SYS0214E - Too many dynamic link modules are attached to program or module.**

Remove some of the DLL dependencies from the program or the module.

☐ **SYS0266E - Specified file was not copied.**

Ensure that the source and target files are correctly named and in the same file system.

☐ **SYS0486E - Reserve parameter not valid.**

Change the reserve parameter to zero (0) and retry the call.

☐ **SYS0528E - Disk did not finish formatting.**

Reboot and try the FORMAT command again.

☐ **SYS0894E - Name cannot be assigned to this partition.**

Check the partition's type and location; make sure you are not trying to name a Boot Manager partition.

☐ **SYS1003E - Command syntax is incorrect.**

Ensure that the appropriate parameters and separators are specified and entered in the proper order.

☐ **SYS1061E - System detected a file error.**

If a read-only file, use the ATTRIB command to change it. If the directory is full, remove unnecessary files and try again.

☐ **SYS1103E - System cannot find the directory specified.**

Ensure that the directory name is correctly specified; create the directory if it does not exist.

☐ **SYS2090E - System cannot load the program.**

Run the program with nothing else running on the system or try loading a backup of the program. If this does not work, increase the value of the RMSIZE= statement in the CONFIG.SYS file, reboot, and try the program again.

☐ **SYS2166E - Printer driver does not exist.**

Ensure that the driver is spelled correctly.

☐ **SYS3171E or SYS3176E - Program encountered a problem and cannot continue.**

Insufficient stack space caused program to end without running exception handlers or an illegal instruction exception was generated. Check the contents of the register to assist you in correcting either of these situations.

Remove Unnecessary Files and Programs

The following files and programs can be deleted from your hard disk to increase your hard disk capacity:

Tutorial Program:

- \OS2\DLL\TUTDLL.DLL
- \OS2\HELP\TUTORIAL.HLP
- \OS2\TUTORIAL.EXE

Game Programs:

- \OS2\APPS\CARDSYM.FON
- \OS2\APPS\JIGSAW.EXE
- \OS2\APPS\KLONDIKE.EXE
- \OS2\APPS\NEKO.EXE
- \OS2\APPS\OS2CHESS.BIN
- \OS2\APPS\OS2CHESS.EXE
- \OS2\APPS\REVERSI.EXE
- \OS2\APPS\SCRAMBLE.EXE
- \OS2\DLL\CHESSAI.DLL
- \OS2\DLL\KLONBGA.DLL
- \OS2\DLL\NEKO.DLL
- \OS2\DLL\SCRAMBLE.DLL
- \OS2\DLL\SCRCATS.DLL
- \OS2\DLL\SCRLOGO.DLL
- \OS2\HELP\JIGSAW.HLP
- \OS2\HELP\KLONDIKE.HLP
- \OS2\HELP\NEKO.HLP
- \OS2\HELP\OS2CHESS.HLP
- \OS2\HELP\REVERSI.HLP
- \OS2\HELP\SCRAMBLE.HLP

Background Bit Maps:

- \OS2\BITMAP\LIGHTHOU.VGA
- \OS2\BITMAP\SWAN.BGA

HPFS Files, if not using HPFS:

- \OS2\CACHE.EXE
- \OS2\DLL\STARTLW.DLL
- \OS2\DLL\UHPFS.DLL
- \OS2\HPFS.IFS

Touch Files, if not using a Touch device:

- \OS2\CALIBRAT.DAT
- \OS2\CALIBRAT.EXE
- \OS2\CALIBRAT.TXT
- \OS2\DLL\FSGRAPH.DLL
- \OS2\DLL\TCP.DLL

- \OS2\DLL\TOUCALLS.DLL
- \OS2\HELP\TCP.HLP
- \OS2\MDOS\VTOUCH.COM
- \OS2\MDOS\VTOUCH.SYS
- \OS2\MDOS\WIN
- \OS2\PDITOU01.SYS
- \OS2\PDITOU02.SYS
- \OS2\SYSTEM\TDD.MSG
- \OS2\SYSTEM\TDDH.MSG
- \OS2\SYSTEM\TDI.MSG
- \OS2\SYSTEM\TDIH.MSG
- \OS2\SYSTEM\TOUCH.DRV
- \OS2\TOUCH.INI
- \OS2\TOUCH.SYS
- \OS2\TOUC021D.BIN
- \OS2\TOUCMOU.BIO

Glossary

This glossary defines only the terms used in this book; it is not a complete glossary for OS/2 2.1.

A

action An operation performed by a computer at the user's request that is applied to an object.

action bar The area at the top of a window that contains options that give users access to available actions.

action choice A selectable action. See also *action.*

active program An application currently running on the computer. See also *application, interactive program, noninteractive program.*

active window The window with which the user is currently interacting. This window has a highlighted border and title bar. See also *inactive window.*

ancestor An object class from which other classes lower in the hierarchy can inherit characteristics and behaviors. See also *class, inheritance hierarchy, parent class.*

APA All points addressable.

API Application programming interface.

application A software program.

application name The complete file specification of an application. See also *application title.*

application title The name of an application as listed on the Desktop. See also *application name.*

ASCII American National Standard Code for Information Interchange.

attribute A characteristic or property of a data type, data structure, object, or file. See also *property, setting*.

AUTOEXEC.BAT A batch file that processes commands that set up the operating system for DOS Sessions.

availability The property of being accessible and usable on demand.

B

back up To copy information onto a diskette or hard disk for record keeping or recovery purposes.

batch file A file that contains a series of commands, which are processed sequentially. It has the extension .CMD or .BAT.

baud rate The speed at which information can travel over a communication line.

bit map A representation in memory of an image.

border A visual indicator of a window's boundaries.

buffer Reserved memory storage used to perform input and output.

button A mechanism used to request or initiate an action. See also *push button, radio button*.

C

cache A high-speed storage buffer used to reduce hard disk access time. Contains frequently accessed information. See also *buffer*.

call A method of transferring control to a specific function or method. See also *function, method*.

cancel An action that removes the active window or menu without processing (changing) it and returns the previous window.

CASE statement Provides the body of a window procedure; there is a CASE statement for every action type message.

case-sensitive A condition in which entries must conform to a specified format (lower-case, uppercase, or mixed-case).

cascaded menu See *conditional cascaded menu.*

CD-ROM High capacity, read-only memory in compact disk form.

character A letter, digit, or other symbol.

check box A control, shaped like a square box, that represents one of several choices. When a choice is selected, a check mark appears in the check box. See also *check mark.*

check mark The symbol used to indicate a selected item in a check box.

child class A subclass of another class. See also *ancestor, class, parent class.*

child window See *secondary window.*

class A set of objects that share common characteristics and behaviors. See also *inheritance hierarchy, parent class.*

class definition file A file that defines a object class and its relationships to other classes. See also *class, object.*

click To press and release a button on the mouse. See also *double-click.*

client application An application that is used to create subclasses or object instances. See also *application, instance, subclass.*

clipboard An area of storage memory that temporarily holds data being passed from one application to another.

command An action that can be performed by a program.

command area Consists of a command entry field and a command prompt. See also *command prompt.*

command line A line where commands can be entered, usually at the bottom of a window. See also *command area.*

command prompt A symbol indicating where to enter commands.

Common User Access (CUA) A set of rules that define the standards and guidelines for user interfaces such as the Workplace Shell.

compiler A program that takes source code and converts it into machine object instructions (also known as *object file* code) that can be read by the hardware platform. The object code is then linked together by a *linker* to build the executable form of the application. See also *linker, object file code.*

compiler options Keywords or switches used to control various aspects of compiling.

conditional cascade menu A submenu that is dependent upon and extends another menu; appears when the arrow to the right of an action choice is selected.

CONFIG.SYS A file added to the root directory during installation that sets the system configuration when the operating system is restarted (booted). See also *configuration, root directory.*

configuration The manner in which hardware and software is organized; describes the options, devices, and applications installed on a system.

container object An object that holds other objects. See also *object.*

context-sensitive help Provides information specific to the field on which the cursor is positioned.

control The method whereby input is given to an application. See also *action, choice.*

CUA See *Common User Access (CUA).*

cursor A symbol associated with an input device that indicates where input will appear on the screen. See also *pointer, text cursor, selection cursor.*

cut To remove a selected object to the clipboard. See also *clipboard.*

D

database A collection of data with a given structure.

data object An object that conveys information such as text or graphics. See also *object.*

DBCS See *double-byte character set (DBCS).*

default An assumed option, attribute, or value. Used when no other option, attribute, or value is specified. See also *default choice.*

default choice A selection provided by the application or system. See also *default value.*

default value A value used when no value is explicitly specified by the user.

dependent See *child class.*

deselect To remove the highlighting from one or more choices. See also *select.*

Desktop An OS/2 2.1 container object that fills an entire window and holds all the object icons. See also i*con, object, window.*

device object An object that enables communication between a computer and another device such as a modem or disk drive. See also *object.*

dialog The exchange of information between a user and a computer.

dialog box See *pop-up menu.*

Dialog Editor A WYSIWYG editor that creates dialog boxes. See also *WYSIWYG.*

direct manipulation The ability to use a pointing device such as a mouse to move objects around a screen. See also *drag and drop, mouse, object.*

directory — A file stored on a hard disk or diskette that contains a grouping of other files, including information such as file size and date the file was created. See also *folder, subdirectory.*

directory tree — A diagram of all directories and subdirectories on a specified drive. See also *directory, subdirectory.*

DOS Session — A collection of system resources that provides a virtualized DOS environment that supports the independent execution of DOS applications.

double-byte character set (DBCS) — A set of characters in which each character is represented by two bytes. Used in languages such as Japanese that contain more characters than can be represented by 256 code combinations.

double-click — To press and release the selection button on a pointing device twice in rapid succession. See also *click, pointing device.*

drag and drop — To move an object, by using a mouse or other pointing device, from one place on the screen to another and then fix the object's position by releasing the selection button of the pointing device. See also *object, pointing device, screen, selection button.*

dynamic data exchange (DDE) — The exchange of data between applications or between an application and a data object. Changes made to data affects both areas. See also *application, data object.*

E

emulate — To imitate another system; the imitating system performs in the same way as the imitated system.

enable — 1. To make functional. 2. Initiate the operation of a device.

enter — To send all selected choices to the computer for processing.

environment variables A series of commands that dictate how a system will operate and what external devices it will recognize. See also *settings*.

event semaphore A signaling mechanism, which enables a thread or process to notify waiting threads or processes that an event (such as termination) has occurred. See also *process, thread*.

extended help Provides information about an entire application window. See also *context-sensitive help*.

F

FAT See *File Allocation Table (FAT)*.

File Allocation Table (FAT) A table used by DOS to allocate disk space for a file. See also *High Performance File System (HPFS)*.

File Manager An OS/2 application that displays files and directories.

file specification A file identifier that includes the file name, extension, path and drive.

flag A file or directory variable that indicates a particular condition is set or not set. See also *hidden flag, setting*.

folder A general-purpose container object used to hold other objects, folders, or applications.

frame A formatted display that contains several visual components specified by the application.

full-screen application An application that occupies the entire screen.

function A routine or process that performs a specific operation such as generating a value. A function is requested or called by an application or system. See also *application, call, method*.

function key A key that, when pressed, initiates a specified sequence of operations.

G

graying The indication that a choice on a pull-down is unavailable.

H

handle A value that represents an object or identifies a file so that the file can be found and opened.

handshaking A method of communication between two pieces of hardware.

hardcopy Physical output from a device such as a printer.

header file A file that contains control information such as language, function-prototype, and data-type definitions. Also known as an *include file*.

help A function that provides assistance and information to the user. See also *context-sensitive help, extended help, keys help*.

help index A selection in the Help menu that displays an alphabetical listing of available help topics.

help window A secondary window that displays information when the user requests help.

hidden flag When set, does not allow a file or directory to be displayed in the directory tree or window. See also *directory tree, flag, hide*.

hide To remove a window from the Desktop. A hidden window can be displayed in the Window List. See also *Desktop, Window List*.

highlight To emphasize a choice by brightening its visual attributes. See also *select*.

High Performance File System (HPFS) An installable file system that uses a cache to provide fast access to large disk volumes. Supports multiple, ac-

tive file systems that have different storage devices. See also *cache, File Allocation Table (FAT).*

HPFS See *High Performance File System (HPFS).*

hypertext links Selectable words or phrases that connect related information. See also *hypergraphic links.*

hypergraphic links Selectable graphics that connect related information.

I

icon A graphical representation of an object. See also *object.*

Icon Editor An OS/2 tool for creating icons.

inactive program An application that is not opened or is opened but not processing. See also *active program.*

inactive window A window that cannot receive input from the mouse or keyboard until it is selected for interaction.

inheritance hierarchy An ordering of object classes based upon relationships between higher and lower class. See also *ancestor, child class, class, parent class.*

install 1. To physically copy files from operating system or program diskettes to specified directories of a hard disk. 2. To add a printer or other device driver to the .INI file.

instance A individual object that belongs to a particular object class. See also *class, object.*

interactive application An active program that is receiving or is ready to receive input from the user. See also *active program.*

interface See *user interface.*

IPF Information Presentation Facility.

J

journal — A special-purpose file that is used to record changes made in the system.

K

keys help — A selection in the Help menu that lists all the key assignments for the current application. See also *help*.

L

LAN — See *Local Area Network (LAN)*.

linker — The part of a compiler that links together object code to build the executable form of an application. See also *compiler, object code.*

list box — A window containing a vertical, scrollable list of selectable objects or action choices. See also *action choice, object, scroll.*

load — 1. To place a diskette into a drive. 2. To move data or applications into memory. See also *install.*

Local Area Network (LAN) — A computer network for local resource sharing.

local device — A directly accessed device. See also *remote device.*

M

Master Help Index — Displays an alphabetical list of all help topics.

maximize — A window-sizing action that increases the window to the largest possible size. See also *hide, minimize.*

menu bar The area immediately below the title bar that contains selectable options. See also *pop-up menu, title bar.*

message 1. A means of communication between OS/2 2.1 and an application. 2. Information displayed by the computer in response to a user action or internal process.

message filter A means of specifying which messages will be handled by the application.

message queue A sequenced collection of messages.

metafile A file that contains the color, shape, and size attributes of a picture or drawing. See also *attribute.*

metalanguage A language used to specify another language.

method A routine or procedure that operates upon an object. See also *function, object.*

migrate To move to an application from one environment to another.

minimize To reduce a window to its smallest possible size (which is an icon) and remove its associated windows from the screen. See also *icon, maximize.*

modem A communications device that enables digital data to be converted into analog signals and then converted back into digital data.

mnemonic A highlighted letter in a menu option that, when typed, selects that option.

mouse A hand-held pointing device. See also *cursor, pointer, pointing device.*

mouse button A mechanism on a mouse that is pressed to select choices or initiate actions. See also *selection button.*

multi-tasking The executing of two or more tasks or applications at the same time.

Multiple Virtual DOS Machine (MVDM)

A system service that coordinates the concurrent operation of multiple DOS sessions. See also *DOS Session.*

N

Network folder

A folder representing a LAN or a group of network objects. See also *Local Area Network (LAN), object.*

notebook

A graphical representation that appears to contain pages separated into sections by tabs; used to simplify the presentation of objects and their data. See also *tab.*

O

object

A logical entity, consisting of a collection of data and methods, that is manipulated within the Workplace Shell. See also *class, method, Workplace Shell (WPS).*

object code

Machine object instructions that can be read by a hardware platform. See also *compiler, linker.*

Object Interface Definition Language (OIDL)

A formal specification language used by the System Object Model to write SOM class definition files. See also *System Object Model (SOM), class definition file.*

object-oriented programming

A method of programming that organizes collections of objects through inheritance relationships and class hierarchy. See also *inheritance hierarchy, System Object Model (SOM).*

OIDL

See *Object Interface Definition Language (OIDL).*

OK

A push button that indicates the acceptance of messages or of changes made to information.

open

To begin working with an object, file, application, or directory.

operating system

Software that controls application processing and provides services for those applications.

P

paint To draw or redraw the contents of a window. See also *refresh.*

panel An arrangement of displayed information in a window.

parent class A class that passes its data and methods to another class immediately beneath it on the inheritance hierarchy. See also *ancestor, class, method, inheritance hierarchy.*

parent window See *primary window.*

partition A fixed-size division of storage on a hard disk.

paste To move the contents of a clipboard into a preselected location in a window. See also *clipboard.*

path A statement that indicates all the directories that must be opened to get to a particular file stored on a particular drive. The directory names are separated by a backslash (\) with the first backslash representing the root directory. The file name and extension is included in the path statement. See also *directory, root directory.*

pointer A symbol on the screen that is moved by a pointing device such as a mouse. See also *cursor, pointing device.*

pointing device A mouse, track ball, joy stick or other device used to move the pointer (cursor) on the screen.

pooling The ability to assign multiple printers to a single printer object.

pop-up menu A menu within a pop-up window that contains a list of actions to be performed on the object associated with that particular pop-up menu. See also *object, pop-up window.*

pop-up window A window that appears on top of another window. See also *pop-up menu, view.*

port A computer connector through which devices can be attached to the computer. See also *device object.*

port designation A 4-character identifier (LPT1 - LPT3 or COM1 - COM3), which is assigned to a device such as a printer or modem. See also *port*.

Presentation Manager The OS/2 Control Program that enables the Workplace Shell to interface with applications and users.

primary window The window in which the main interaction between users and the application takes place. In OS/2 2.1, each application starts in its own primary window. See also *secondary window*.

print job Information that is printed or waiting to be printed. See also *queue*.

Print Manager Part of the spooler that manipulates print jobs and enables print queues to be viewed. See also *print job, queue*.

process An instance of an application that owns system resources such as threads. A process consists of one or more threads. See also *application, event semaphore, system resource, thread.*

program name See *application name*.

program title See *application title*.

property An attribute generally used to refer to the setup of a printing device or job. See also *attribute, device, setting*.

protocol A set of rules controlling the communication and transfer of data between two or more processes, applications, or devices. See also *process*.

pull-down menu An extension of the menu bar that drops down a list of available action choices when an option in the menu bar is selected. See also *action choice, object, menu bar.*

push button A control that displays text/graphics in a rounded-corner rectangle, and which invokes an immediate action when selected.

Q

qualifier A modifier used to ensure a name will be unique.

queue A list of objects waiting to be processed, such as a list of print jobs waiting to be printed.

R

RAM Random access memory.

refresh To update the information in a window. See also *paint, restore*.

remote device A device that is accessed through a telecommunication line.

resource An object, file, application, or device needed to perform required system operations.

resource file Contains definitions of fonts, templates, accelerators, and mnemonics.

restore To return a window to its original size or position. See also *maximize, minimize*.

root directory The first directory on a drive, which contains all other subdirectories and files. See also *path, subdirectory*.

S

SAA See *Systems Application Architecture (SAA)*.

screen The physical surface of a monitor upon which information is presented to users.

scroll To move information vertically or horizontally so that it can be seen in the display window; new data appears at one edge while existing data disappears at the opposite edge.

scroll bar A control, horizontally or vertically aligned, that enables a user to scroll data. The scroll bar can be used only with a mouse. See also *mouse*.

secondary window A window associated with the primary window. See also *primary window*.

select To mark (highlight) or choose an item, which then must be entered to be processed. See also *enter, highlight, selection button*.

selection button Button one on a mouse, which is pressed to select a menu choice. See also *mouse*.

selection cursor A cursor used to indicate (by highlighting) a choice or entry field selected by the user. See also *highlight*.

separator 1. A boundary that divides two adjacent areas. 2. A character (such as "\") used to delimit character strings.

server A computer or workstation on a local area network (LAN) that shares its resources with other computers or workstations on the network. See *Local Area Network (LAN), workstation*.

session An instance of an opened application or command prompt. The session types in OS/2 2.1 are OS/2 window, OS/2 full screen, WIN-OS/2 full screen, WIN-OS/2 window, DOS window, and DOS full screen. See also *DOS Session, MVDM*.

setting A characteristic of an object that can be changed or modified.

Settings An option in a pop-up window of an object that, when selected, displays the settings of that object. See also *object, setting*.

shadow A duplicate of an object. If a change occurs, it is reflected in both the object and its shadow. See also *object*.

shared resources Data, applications, directories, and devices made available for use on an operating system or network. See also *network, operating system*.

shutdown In OS/2 2.1, the procedure required before the computer is powered off that ensures that data and configuration data is not lost. See also *configuration.*

SOM See *System Object Model (SOM).*

spooler Stores jobs that are waiting for an available printer or port. An application writes data to a disk until it is able to be sent to a printer device; it prevents the data from being intermixed with data from other sources. See also *device, Print Manager, spooling.*

spooling The process of freeing system resources by temporarily storing print jobs while waiting for an available printer or port.

start up To begin the operation of an application. See also *open.*

standard window A CUA-defined window. See also *Common User Access (CUA).*

subclass See *child class.*

System Menu In OS/2 2.1, the pull-down menu available from the System Menu icon, which enables a window to be moved, sized, minimized, maximized, and restored. See also *icon, maximize, minimize, pull-down menu, restore.*

System Object Model (SOM) A language-independent, object-oriented programming tool that allows objects to be created, defined, changed, and manipulated. See also *object-oriented programming, Workplace Shell (WPS).*

system resource An item or component that provides information used in the system.

Systems Application Architecture (SAA) A formal set of rules that enables applications to be run in different computer environments.

T

tab A graphical representation of a notebook tab; when selected, turns the notebook page. See also *notebook.*

tag A markup language word and its attributes.

template An object model used to create other objects.

text cursor A cursor that indicates where typed input will appear.

thread An executable unit within a process. See also *process.*

title bar The area at the top of a window that contains the window title, title bar icon, and minimize, maximize, and restore buttons. See also *title-bar icon.*

title-bar icon The small icon in the upper-left corner of the title bar that represents the object opened in the window.

Tree An OS/2 window in the File Manager that indicates the organization of drives and directories. See also *directory tree.*

U

Using help A selection in the Help menu that provides information about the help function. See also *help.*

user interface The hardware/software that enables a user to interact with and perform operations on a computer.

V

value A quantity assigned to a variable, constant, or parameter.

view A way of looking at an object. See also *object.*

virtual memory (VM) Addressable space that appears to be processor storage space; does not have a fixed physical location.

W

window A rectangular area of the screen. Windows can be smaller than or equal in size to the screen; they can also overlap other windows.

Window List A pop-up window that lists all open windows. Used to view hidden windows, switch from one application to another, close an application, etc. See also *hide, open, pop-up window.*

WIN-OS/2 A full-screen or windowed session that supports Window applications. See also *session.*

working directory A specified directory that becomes the current directory when an application is opened. See also *directory.*

work area A folder setting that groups objects together for a specific task.

Workplace Shell The object-oriented user interface component of OS/2 2.1; used to manage applications installed and running under OS/2 2.1. See also *object-oriented programming, System Object Model (SOM).*

workstation An area of work that usually contains a monitor (display screen), keyboard, and access to a printer or other copy device.

WPS See *Workplace Shell.*

WYSIWYG What You See Is What You Get. Displays on the screen the way text will look when formatted on a printer.

Index

A

accelerator keys 402
accessing
  data 114
  disk partitions 27
  DLL functions 39
  methods and data 110
  object 90
  objects through SOM 99
  online tools 178
  spooler object 59
  system memory 48
action choices, defining 129
adding
  applications to Desktop 136
  device drivers 27
  environment variables 30
  help 199
  items to menus 128
  memory 12
  menu items 128
  options to pop-up menu 127
  page to Settings notebook 121
  TCP/IP directories 152
addresses, types of 143
APPEND command 41
applications
  16- and 32-bit models 46
  accessing system memory 48
  adding help to 199
  backing up DOS and Windows 9
  building new 156
  building OS/2 2.1 209
  client 105
  compiling and optimizing 34
  creating 87
  creating 32-bit executable 33
  creating PM 157
  design tips for 401
  DOS security device 78
  event-oriented 155
  header files for 373
  linked with run-time libraries 38
  Magic Window 157
  migrating 27, 136
  object-oriented 100
  printing 59
  removing unnecessary 406
  reusable submodules for 361
  running installation 135
  sending message to 196
  server communication 150
  setting up icons for 139
  starting OS/2 196
  TCP/IP 143
  Warehouse Inventory 209
attribute
  auto 197
  compact 184
  controls= 191
  ctrlarea= 192
  database= 196
  global 194
  hide 189
  inform 196
  instance 120
  launch 196
  page 191
  printer driver extended 68
  reftype= 197
  res= 184
  tag 184
  viewing 90

B

backing up applications 9
binding files 109
Boot Manager
  Boot Manager Menu 20
  disk space requirements 13
  installing 13
  placement of 16
  setting as Startable 18
  setting timer 16

setting up partitions for 9
Borland C++
calling convention options 49
calling conventions 45
resource compiler 33
source code 50
build command directive 51

C

C Set/2
calling convention options 48
calling conventions 43
installing 30
listing options 49
source code options 49
calling conventions
Borland C++ 45
Borland C++ options 49
C Set/2 43
C Set/2 options 48
category of OS/2 functions
See functions by category
checklist, error 204
class
child 102
creating SOM object 105
definition file 105, 119
implementer file extensions 109
implied 107
library, building 110
metaclass 101
method 99
method, invoking 114
object 99
object, defining 104
object, example of 104
parent 99
root object (SOMObject) 100
socket 146
SOM 118
WPS 118
WPS storage 119
class definition file, example 108
class-specific macros 111
Clipboard 7
command
APPEND 41
COPY 68, 81
macro 52
PRINT 81
SET 180
VIEW 178, 180
Common User Access
interface 87
visual components 97
window types 95
communicating
among machines 141
with windows 162
compiler
Borland C++ 33
C Set/2 30
C-language 110
calling convention options 48
definition of 33
IPF 179
IPF control words 187
listing options 49
multipass 34
SOM 105, 109
source code options 49-50
using inference rules 51
working directory 42
compiling
and linking 36
applications 34
errors 189
TCP/IP applications 152
composed view 90
concatenating online files 180
conditional cascade menus 128
CONFIG.SYS file
adding device driver statements to 80
automatically updating 30
environment variables 40
modifying 28
sample of 40
configuration
modifying system 23
WIN-OS/2 printer 69
container object 88

contents view
  details view 92
  icon view 92
control word
  .im 183
  using 187
COPY command 68, 81
coverpage window 190
creating
  32-bit executable 33
  applications 87
  class definition file 105
  customized Desktop 133
  development environment 40
  dialog template 54
  DLL 109
  footnotes 198
  help 134, 177
  help instance 202
  HELPINIT structure 201
  hypergraphic links 197
  hypertext links 195
  import libraries 39
  main module template 157
  Master Help Index entries 193
  menu bar 200
  object instances 131, 133, 135
  object shadows 129
  objects using SOM 99
  partition for DOS 17
  printer objects 62, 69
  SOM object classes 105
  split windows 197
  standard window 180
  symbols 187
  window titles 188
Crtl+Z processing 81
CSETENV.CMD 30
CUA
  See Common User Access
customizing installation 25

D

daemon 146
data
  accessing 114
  attaching 164
  global 162
  global variable declarations 37
  losing 14
  object 88
  passing 164
  passing to a dialog procedure 216
  restoring 27
  retrieving 225
  saving 127
  section 107
  standard and raw 61
  structures 122, 130
  viewing 90
debugger
  Cross Net (XNET) 151
  installed by C Set/2 30
  restrictions 34
debugging
  code 37
  in SOM 113
default
  control area 192
  database 137
  device font 81
  instance attributes 120
  job properties 73, 80
  listing options 49
  port value 78
  print priority setting 61
  printer driver 67
  printer object 64
  procedure 161, 225
  settings 23
  status of source code options 49
  values 124
definition list 185
deleting
  files 94
  partitions 14
  print job 65-66
  printer driver 70
  SWAPPER.DAT 27
  unnecessary files and programs 406
Desktop

changing objects on 132
changing system configuration through 23
setting passwords through 402
details view 92
Dev functions 233
device drivers
adding 27
port 79
support 3
support for SVGA 11
Windows 3.1 4
device object 88
device-independent files 61
dialog
box template 200
boxes, separating 54
procedure 225
procedure styles 160
procedure, passing data to 216
Dialog Editor 53
direct manipulation 94
direct printing mode 60
directory tree
IBM C Set/2 30
OS/2 2.x Toolkit 29
display errors 206
DLL
See dynamic link library (DLL)
DOS
creating a primary partition for 17
object, copying 136
security device applications 78
Settings 137
Settings, location of 5
Dos functions 235
drag and drop
icons 139
printing using 81-82
template 132
using 123
drawing column-heading text 224
dynamic link library (DLL)
building resources into 53
creating 109
dynamically linked run-time libraries 38
modifying routines in 37
module definition file 133
structures defined in 199
using 36

E

editor
Dialog 54
Icon 53, 139
System 67
encapsulation 102
enhancing performance
using object-oriented model 88
using optimizing 34
using static linking 39
using wpSaveDeferred 126
environment variable
adding 30
adding directories to 152
CONFIG.SYS 178
definition of 40
DPATH 41
HELP 31
INCLUDE 42
LIB 42
PATH 41
setting 180
TMP 42
errors
checklist 204
compatibility 87
compiling 42, 189
display 206
footnote 206
hypergraphic link 205
hypertext link 205
indexing 204
initialization 78
memory 12
message 403
miscellaneous 206
Printer Overrun 77
search 205
SOM 114
spooler 59
tagging 204

example
  class definition file 108
  device drivers 79
  help source file 201
  Magic Window make file 166
  metafile format 66
  method stub 110
  module window procedure 161
  partitioning 19
  raw data format 67
  RPC 149-150
  socket 147
  SOM object classes 104
  tagging 186, 190, 193, 195, 198
  traditional window procedure 160
  Warehouse Inventory main module 212
  WPS inheritance hierarchy 118
executable program
  creating 33
  linked with module definition file 56
  reducing size with DLLs 36
  statically linked run-time libraries 38
  using NMAKE 50
External Data Representation (XDR) 149, 151

F

fast system fonts 76
FAT
  See File Allocation Table (FAT)
features
  advanced 33
  DOS Settings 137
  drag and drop 94, 123
  help 178
  installable 24
  pooled printing 63
  SOM programming 102
  using list box 223
features of OS/2 2.1
  See OS/2 2.1 features
File Allocation Table (FAT) 21
files
  accessing 13
  ASCII 81
  backing up 9
  batch or .CMD sample 41
  binding 105, 109
  class definition 105
  concatenating online 180
  creating make 166
  deleting 94
  device-independent 61
  generating with SOM compiler 109
  header 373
  imbedded 187
  implementer extensions 109
  make 51
  Master 183
  module definition 56
  removing unnecessary 406
  resource script 53
  REXX-language batch 135
  symbol 46
  types of 82
flat memory model 4, 11, 47
folder
  Command Prompts 6, 136
  for migrated applications 139
  Games 7
  IBM C Set/2 30
  OS/2 System 7, 59
  predefined system 134
  Productivity 7
  Spooler 60
  System Setup 82
  Templates 62, 69, 131-132, 135
font
  default device 81
  fast system 76
  installing new 76
  soft 75
footnote
  creating 198
  errors 206
format
  address types 144
  classname method 112
  inference rules 51
  metafile 66, 82
  PM_Q_RAW and PM_Q_STD 61
  printer-specific 82

raw data 67
setup string 124
formatting
backing up applications before 9
hard drive 10
partitions 25
remaining drives 27
when setting up HPFS 11
function
attaching data 165
class method 99
inlining 36
keys 402
model types 156
modules, using 156
object class 133
prototypes 37
query 166
replaceable SOM 113
returning FALSE 161
reuseable 157
REXX 135
statically linked 37
using thunking to convert 47
WPS 132
wrappers 155
functions by category
Dev 233
Dos 235
Gpi 262
Prf 307
Win 308

G

Gpi functions 262
graphical user interface 87

H

hard disk
deleting unnecessary files 406
saving space 112
space requirements 13
header file
See also include file
definitions 201
implementation (.IH) 110
Magic Window, list of 373
MAIN.H 160
syntax 37
using 37
Warehouse Inventory, list of 373
headings
boldface 186
Contents window 189
help
adding 199
coverpage window 190
creating 134
instance, creating 202
Keys 182
Keys Help 402
Master Index entries 193
online 8
Online Help program 30
online menu 181
subtables 200
tables, setting up 199
view 93
HELPINIT structure, creating 201
High Performance File System (HPFS)
formatting for 11
installing 21
highlighting online text 187
HP-GL/2 77
HPFS
See High Performance File System (HPFS)
hypergraphic link
creating 197
errors 205
hypertext link
creating 195
errors 205

I

icon
changing 126
object representation 6
opening application with 87
Printer 62

printer driver 70
setting up 139
view 92
Icon Editor 53, 139
ID
object 133
OBJECTID 126
resource 196
SOM 112
identifier
definitions 160
menu 201
res= 196
unique 56
window 184
WM_USER 163
IMPLIB.EXE 39
import library 39
in-use list 122
include file
See also header file
used by several source files 46
index
Master Help 193
primary entry 192
secondary entry 193
indexing errors 204
indirect manipulation 94
inference rules format 51
Information Presentation Facility (IPF)
compiler 179
control words 187
creating help with 177
help instances 134
indexing tags 192
library 199
special characters 187
tags 183
inheritance 102
installation programs, running 135
installing
Boot Manager 13
C Set/2 compiler 30
from a CD-ROM 9
from a local area network 9
HPFS 21
memory requirements for 22
MMPM/2 11
OS/2 2.1 9, 13, 22
OS/2 2.x Toolkit 28
printer driver 70
questions on 10
queue driver 80
recommended sequence 9
WorkFrame/2 28
WPS object 134
interface
controls 93
CUA 87
CUA visual components 97
definition (.SC) file 108
graphical user (GUI) 177
object-oriented user 104
windowed user 136
WPS programming 117
international language support 179
internet addressing and domains 143
Internet Protocol (IP) 142
intrinsic inlining 36
item types 122

J

job
contents, previewing 66
details view 64
icon view 64
status, previewing 65
Job Contents View utility 66

K

key
accelerator 402
print screen 82
keyboard, locking 402
KEYNAME values 124, 133
keyword
ALL 52
describing function definitions 45
module definition file 57
rcinclude 54

L

LAN
  See local area network
LAN independent shell 7
Large Buffers option 77
libraries
  class 101
  dynamic link 36
  help 193, 195
  import 39
  IPF 199
  run-time 38
  SOM class 110
LINK386.EXE 28, 30
linker
  invoking 36
  map file 49
  OS/2 (LINK386.EXE) 41
  using import libraries 39
linking TCP/IP applications 152
links
  hypergraphic 197
  hypertext 195
list
  definition 185
  ordered 184
list box
  aligning columns 223
  messages 224
local area network (LAN) 7, 9, 61
locking keyboard and mouse 402
logical drive
  accessing files on 13
  for swapper file 26
  setting up 22

M

macros
  $* and $** 52
  debugging 113
  in make file 167
  NAMEIT 187
  SOM 111
Magic Window
  header files 373
  main module 157
  make file 166
  submodules 168
  using 157
main modules
  differences between 211
  template 157
  Warehouse Inventory 211
make file
  creating 166
  definition of 51
  sample of 52
MAKEINI utility 133
Master file 183, 187
Master Help Index 193
memory
  adding 12
  allocating 48, 122
  errors 12
  flat memory model 4, 11
  freeing 123
  freeing cache 133
  minimum for installing OS/2 2.1 22
  recommended amount for system 12
  reduction using dynamic linking 39
  system 11-12
memory contention 12
menu
  adding options to 127
  bar, creating 200
  class-specific items 128
  conditional cascade 127-128
  job icon view pop-up 65
  online help 181
  pop-up 127
  Printer Properties 75
message
  error 403
  for list box 223
  processing requirements 225
  queue 155
  sending to application 196
  sending to window 163
  SOMError 114
  WM_DRAWITEM 224

WM_MEASUREITEM 224
metaclass, definition of 101
method
accessing 110
Direct Manipulation 123
invoking SOM 114
Object Information 131
Object Usage 122
override 103
Pop-Up Menu 127, 129
Save\Restore State 126
section 107
Settings Notebook 121
Setup 124
tracing 112
using pointers in 114
using SOM 111
WPS class 120
microprocessors 11
migrating
applications from Desktop 136
applications from other operating systems 27
MMPM/2 support 4
mode
automatic mixed 75
emulation 72
transparent 76
working 75
modular approach 155
module definition file
keyword 57
using 56
modules
building applications with 209
generic and application-specific 229
list of function 210
list of function collection 211
list of procedure 210
list of reusable 361
Magic Window 168
Magic Window main module 157
Magic Window make file 166
using function 156
Warehourse Inventory 210
Warehouse Inventory 229
Warehouse Inventory main window 216
Warehouse Inventory main module 211
mouse, locking 402
Multiple Virtual DOS Machine (MVDM) 5, 41

N

NAMEIT macro 187
namesavers, using 145
network
printer, spooling to 60
protocols 141
server 61, 63
TCP/IP 146
NMAKE utility 50
NMAKE.EXE 41

O

object
behavior, defining 131
class relationships 99
classes, defining 104
classes, example of 104
classifying 115
code, generating 109
copying DOS 136
definition of 99
determining choices 115
differences bewteen 117
encapsulation 102
example of relationships between 101
information, obtaining 131
inheritance 102
installing WPS 134
metaclass 101
methods 122
nonpersistent 119
persistent 119, 126
print job 64
print-related 64
printer 62
printer driver 67
printer port 77
queue driver 80
registering 132
removing 123

represented by icon 92
retrieving information 113
selecting 93
shadow 92, 129
SOMObject 100
spooler 59
templates, using 131
types 88
viewing 90
OEM driver, installing 70
online
files, concatenating 180
help features 178
help menu 181
information 177
text, highlighting 187
tools 178
online help
See help
optimized window 95
optimizing
applications 34
options for Borland C++ 35
options for C Set/2 35
options
Borland C++ calling convention 49
Borland C++ optimization 35
Borland C++ source code options 50
C Set/2 calling convention 48
C Set/2 listing 49
C Set/2 optimization 35
C Set/2 source code 49
print performance 76
OS/2 2.1
C-language binding files 105
creating a partition for 18
CUA component 87
drag and drop 123
hard disk space required 12
help and online programs 177
installing 13, 22
list of alphabetized functions 233
Presentation Manager 155
print subsystem 59
printer drivers 71
REXX 135
setting passwords and screen savers 402
System Object Model (SOM) 99
TCP/IP 142
thunking subsystem 47
Toolkit 28
Workplace Shell 117
OS/2 2.1 features
32-bit flat memory model 4
32-bit graphical environment 5
additional device driver support 3
CD-ROM installation 3
Clipboard 7
Command Prompts folder 6
dual thread support 4
improved online help 8
integrated LAN resources 7
MMPM/2 support 4
MVDM 5
OS/2 Desktop 6
OS/2 System folder 7
Settings notebook 7
thunking 5
Windows 3.1 support 4
Workplace Shell 5
OS/2 2.x Toolkit
linker and compiler 33
NMAKE utility 50
OS/2 functions by category
See functions by category
OS/2 System 7, 11, 23, 59, 137
OS2.INI 133

P

page
adding 121
attribute 191
General 120, 122, 132
modifying 120
Printer driver 73
Program 136
removing 122
Sessions 137
paging out process 12
partition
creating OS/2 2.1 18

deleting 14
extended 13
formatting 25
HPFS-formatted 21
placing DOS in a primary 17
primary 13
recommended MB for OS/2 10
partitioning example 19
passwords, setting 402
performance
See enhancing performance
PICVIEW tool 66, 82
PM_Q_RAW 61, 67, 80
PM_Q_STD 61, 66
PMPLOT queue driver 80
polymorphism 103
pooled printing 63
pop-up
See menu
pop-up window
See window
port
device drivers 79
output, selecting 78
serial 79
Presentation Manager (PM) 5, 117, 155
Prf functions 307
primary window 95, 197
print
job object 64
job status, previewing 65
priority, setting 61
screen key 82
subsystem 59, 82
PRINT command 81
print performance options
fast system fonts 76
HP-GL/2 77
large buffers 77
printer patterns 77
printer
object 62
object, creating 69
port object 77
printer driver
extended attribute 68
installing 70
installing OEM 70
object 67
provided with OS/2 2.1 71
printing
binary file 81
direct 60
opened window 83
pooled and shared 63
queued 61
using command line 81
using drag and drop 82
problems
compiling 42
debugging 34, 37
installing 31
installing printer drivers 69
moving source code 38
printer 66
printing 77
Program page 136
programming
construct 117
features of SOM 102
interface 117
object-oriented 99
using modular approach 155
programs
See applications
properties
deciding which to use 74
job 67
printer 67
setting 73
protocols
fundamental TCP/IP 142
Internet (IP) 142
less common 151
network 141
TCP/IP 143, 152
Transmission Control (TCP) 143
User Datagram (UDP) 143
push button
Create 69
creating 189
help 178

Job properties 73
Open 76
specifying 191

Q

queue
driver object 80
message 155
print 63
printer 60
queued printing 61

R

raw data spooling 61
remote procedure call (RPC)
example 149-150
levels 148
using 146
requirements
See also restrictions
creating executable program 33
disk space for Boot Manager 13
first physical disk drive 13
hard disk space 12
HPFS setup 11
initiating DOS from Drive C 17
memory for installing OS/2 2.1 22
microprocessor 11
procedure 161, 225
spooler space 59
static linking 39
system memory 11
unique identifiers 56
resource compiler
See also compiler
for PM-based applications 33
invoking 53
resource script file
identifiers 56
INI.RC 133
structures defined in 199
using 53
resources
maintaining 53
tracking 122
WPS 130
restrictions
See also requirements
ANSI compliance 42
calling conventions 44
data area 164
debugger 34
dialog box 203
HPFS-formatted partitions 21
installing C Set/2 30
installing OS/2 2.x Toolkit 30
LIB file and path 42
moving swapper file 26
printer driver processing 68
REXX batch files 135
routers, using 145
run-time library
linked with applications 38
SOM 99
using 38

S

saving
compilation time 107
data 127
disk space 23, 112
resources and memory 34
time 233
screen
See window
screen savers, setting 402
secondary window 95, 197
Sessions page 137
SET command 180
setting
a partition installable 18
Boot Manager timer 16
job properties 73
passwords and screen savers 402
print priority 61
queue options 80
setting up
development environment 152
environment properly 31

Help tables 199
HPFS 11
icons 139
partitions 18
system as logical drives 22
Settings notebook
DOS object 136
Lockup 402
modifying pages in 120
Print screen tab 82
Printer driver page 73
printer object 63, 67, 77
Queue options tab 80
spooler object 60
template checkbox in 132
Workplace Shell control 7
settings view 90
shadow object 92, 129
shared printing 63
socket
classes 146
example 147
using 145
SOM
See System Object Model (SOM)
SOMObject 100
SOMClass 101
SOMClassMgr 101
special characters, using 187
split window
creating 197
help 178
SPOOL directory 60
Spooler folder 60
spooler object 59
spooling standard data 61
standard window 95
storage
avoiding fragmentation 16
classes 119
saving memory 34
stub procedure, definition of 110
subclass
See class
submodules
See modules
support
16- and 32-bit application 46
calling conventions 42
development 87
device drivers 3, 11
dual thread 4
international language 179
MMPM/2 4
Windows 3.1 4
swapper file
hard disk space required 12
moving after installation 27
moving during installation 26
relocating 26
SWAPPER.DAT 12, 27
swapping
4KB pages 48
memory pages 12
symbol file 46
symbols, creating 187
syntax
See also format
compiler 179
compiling 179
Tag language 183
System Editor 67
system font metrics 224
System Object Model (SOM)
class libraries 110
classes 118
compiler 109
creating object classes 105
debugging 113
IDs 112
inheritance 102
object 117
polymorphism (method override) 103
replaceable functions 113
run-time environment 103
SOMClass 101
SOMObject 100
understanding 99
using methods and macros 111
System Setup 11, 23, 82, 137

T

tab-stop positions, adjusting 224
tag
  :artwork. 197
  :ctrl. 191
  :dl. 185
  :font. 188
  :h1. 184
  :link. 195
  :ol. 184
  :sl. 184
  :title. 188
  attributes, using 184
  caution 186
  heading 188
  indexing 192
  IPF 183
  note 186
  paragraph 183
  text string 183
  userdoc 183
  warning 186
Tag language
  syntax 183
  using 178
tagging errors 204
TCP/IP
  directories, adding 152
  Programmer's Toolkit 142
  protocol suite 142
  protocols 152
  protocols and applications 143
  using 141, 145
template
  C file 110
  definition of 132
  dialog 54
  dialog box 200
  object 131
  printer 62
  printer object 69
  source module 157
  Warehouse Inventory 209
thunking
  interfacing 16- and 32-bit 5
  process 47
timer
  changing settings 21
  setting 16
tips
  coding 107
  compiling 42
  debugging 37
  design 401
  header file 43
  heading and index 194
  installing 22
  installing OS/2 2.1 9
  memory 12
  printer driver 72
Toolkit
  See OS/2 2.x Toolkit
Transmission Control Protocol (TCP) 143
Trap 2 errors 12

U

USEITEM 122
User Datagram Protocol (UDP) 143
utility
  IMPLIB 39
  Job Contents View 66
  MAKEINI 133
  NMAKE 50
  OS/2 print 81
  SOM 99
  WPS Class List Object 134, 136

V

value
  KEYNAME 133
  timeout 78
variables, KEYNAME 124
view
  closing 122
  composed 90
  contents 92
  details 130
  help 93
  job details and job icon 64

open 129
opening 122
public and private 102
selecting object 115
settings 90
VIEW command 178, 180

W

Warehouse Inventory
application 209
dialog procedure 225
header files 373
main window procedure 216
main module 211
submodules 229
warning
heading 189
installing DOS 19
IPF compiler 183
SWAPPER.DAT 27
Win functions 308
window 13
Add Programs 138
Advanced Options 27
attaching data to 164
codepage 180
communicating with PM 162
container control 123
Contents 188
coverpage 190
Create a Printer 69
CUA types 95
DOS Configuration 25
FDISK 14
Find Programs 137
Fixed Disk Utility 21
footnote pop-up 198
headings 189
Installation Drive Selection 13
IPF 177
Job Properties 80
Location of Boot Manager pop-up 15
Location of Partition pop-up 17
Migrate Programs 138
New Name pop-up 18
OS/2 Configuration 25
OS/2 Setup and Installation 22
OS/2 Setup and Installation (2nd) 24
printing a 83
procedure styles 160
Select Printer(s) 24
Select the File System 21
Selected Programs 138
sending message to 163
Size of Partition pop-up 16
split 197
standard, creating 180
Startup Values 16
System Configuration 23
titles, creating 188
Type of Partition pop-up 17
Warning 14
Windows 3.1 support 4
WorkFrame/2
installing 28
Toolkit tool entries 29
Workplace Shell (WPS)
and CUA 87
class registration 136
classes 118
feature of OS/2 5
implied metaclasses 119
in-use list 122
object behavior 131
object classes 120
object, installing 134
objects, creating help for 134
predefined open views 129
programming interface 117
storage classes 119
using 117
using objects in 132
WPObject 119
WPS Class List Object utility 134

WHY WASTE VALUABLE TIME...



WHEN YOU CAN GO STRAIGHT TO THE SOURCE?



Introducing the only magazine exclusively devoted to software developers working in the OS/2 environment.

- ❒ **THE SOURCE for OS/2 tips and techniques, direct from Big Blue to you.**
- ❒ **THE SOURCE for valuable news and information for OS/2 application developers.**
- ❒ **THE SOURCE for those working with everything from LANs to multimedia to DBMS architecture.**
- ❒ **THE SOURCE for new software development ideas and products.**

Subscribe now and pay just $39.95 for four big issues, delivered every single quarter to your office desk – straight from the source.

❒ YES! Send me a full year of IBM OS/2 Developer. I'll pay just $39.95 for four big issues!

Name_____ Title_____

Company_____

Address_____

City/State/Zip_____

Charge My Credit Card : ❒ VISA ❒ MC ❒ AMEX

Card Number: _____

Expiration Date: _____ Signature: _____

You may pre-pay by enclosing a check or money order made out to IBM OS/2 Developer, or by enclosing VISA, MasterCard or American Express number, expiration date, and your signature. Delivery of first issue will be in 6-8 weeks. Canadian and international surface mail orders, add $16 per year for postage. International surface mail is an additional $30 per year for postage. Checks must be in U.S. funds.

SPEED YOUR ORDER!
1-800-WANT-Os2
1-708-647-5960
(OUTSIDE THE U.S.)

FAX:
1-708-647-0537

MAIL THIS FORM TO:
IBM OS/2 DEVELOPER
P.O. BOX 1079
SKOKIE, IL, 60606

VNR Computer Library

OS/2®/Programming

OS/2® 2.1 CORPORATE PROGRAMMER'S HANDBOOK

Nora Scholin, Martin Sullivan, and Robin Scragg

Written by OS/2 insiders, the *OS/2® 2.1 Corporate Programmer's Handbook* makes it easy for you to write programs under OS/2 2.1. Using the tips and procedures included in this book, you will be able to immediately write and implement useful new applications, even if you are a first-time user of OS/2!

The Handbook is divided into four sections that:

- Contain tips on installing and using OS/2 2.1.
- Explain the important design concepts as they apply to writing a new application.
- Show you how OS/2 applications can be assembled using the modular approach to coding. (Real-world code samples are provided that are typical of, but not exclusive to, corporate usage.)
- Provide a reference section that lists OS/2 functions by category.

The enclosed diskette contains the code and functions used in the book, which you can modify to build your own OS/2 applications.

About the Authors

Nora Scholin is a technical writer and editor who authored three device driver reference books for IBM's OS/2 Technical Library and IBM's new OS/2 2.1 Command Reference.

Martin Sullivan is a consultant for IBM and a member of the OS/2 development team, specializing in display drivers for the OS/2 operating system.

Robin Scragg was a member of the IBM team that installed and tested updates to OS/2 2.0.

OS/2 is a registered trademark of IBM Corporation.
OS/2 accredited logo is a trademark of IBM Corp. and is used by Van Nostrand Reinhold under license.

Cover art by Nora Scholin, design by J. Herder

ISBN 0-442-01598-4

90000

9 780442 015985

Van Nostrand Reinhold
115 Fifth Avenue, New York, NY 10003